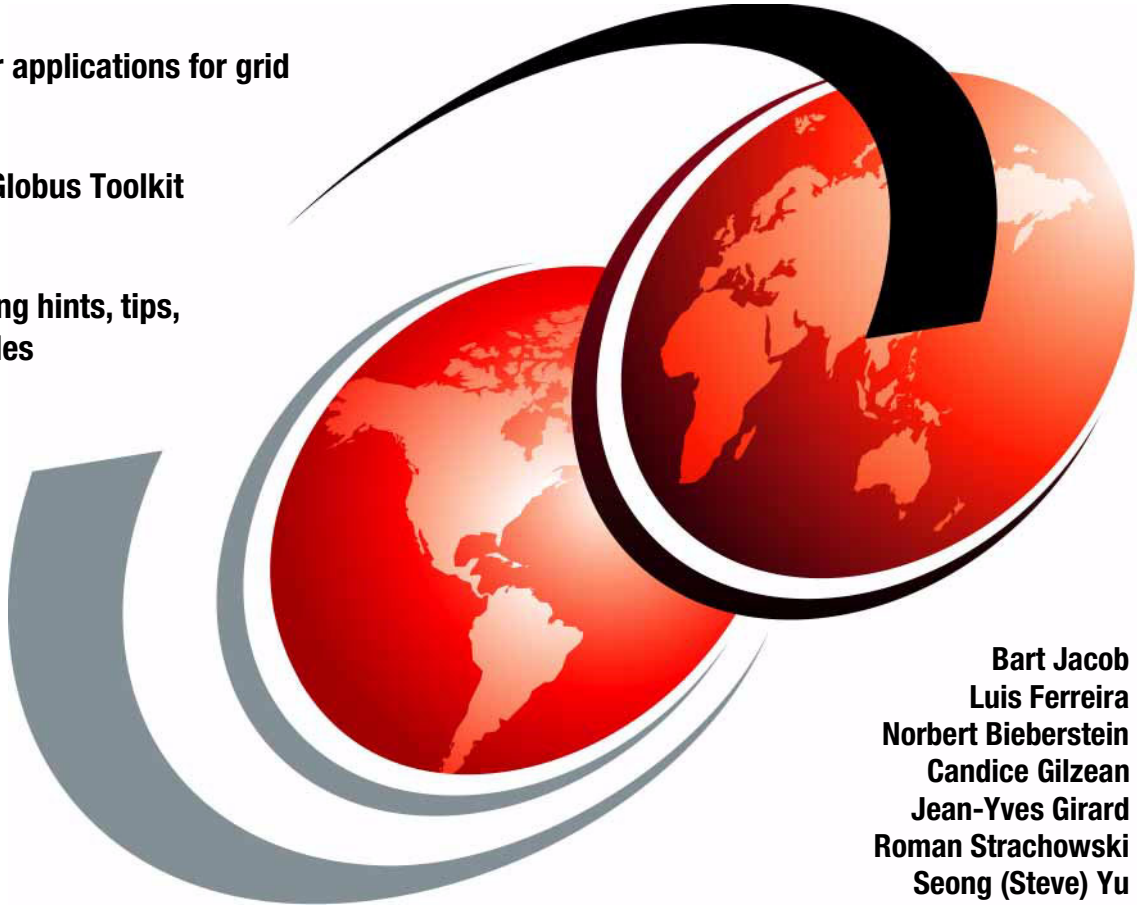
IBM

Enabling Applications for Grid Computing with Globus

- Enable your applications for grid computing
- Utilize the Globus Toolkit
- Programming hints, tips, and examples



Bart Jacob
Luis Ferreira
Norbert Bieberstein
Candice Gilzean
Jean-Yves Girard
Roman Strachowski
Seong (Steve) Yu

Redbooks

ibm.com/redbooks

International Technical Support Organization

Enabling Applications for Grid Computing with Globus

June 2003

SG24-6936-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (June 2003)

This edition applies to Version 2.2.4 of the Globus Toolkit.

© Copyright International Business Machines Corporation 2003. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Notices xi
Trademarks xii

Preface xiii
The team that wrote this redbook xiii
Become a published author xvi
Comments welcome xvi

Chapter 1. Introduction 1
1.1 High-level overview of grid computing 3
1.1.1 Types of grids 3
1.2 Globus Project 4
1.2.1 Globus Toolkit Version 2.2 5
1.2.2 OGSA and Globus Toolkit V3 5
1.3 Grid components: A high-level perspective 6
1.3.1 Portal - User interface 6
1.3.2 Security 7
1.3.3 Broker 8
1.3.4 Scheduler 9
1.3.5 Data management 9
1.3.6 Job and resource management 10
1.3.7 Other 11
1.4 Job flow in a grid environment 11
1.5 Summary 11

Chapter 2. Grid infrastructure considerations 13
2.1 Grid infrastructure components 14
2.1.1 Security 14
2.1.2 Resource management 17
2.1.3 Information services 22
2.1.4 Data management 25
2.1.5 Scheduler 29
2.1.6 Load balancing 31
2.1.7 Broker 33
2.1.8 Inter-process communications (IPC) 33
2.1.9 Portal 34
2.2 Non-functional requirements 35

© Copyright IBM Corp. 2003. All rights reserved.

2.2.1 Performance 36
2.2.2 Reliability 37
2.2.3 Topology considerations 38
2.2.4 Mixed platform environments 40
2.3 Summary 41

Chapter 3. Application architecture considerations 43
3.1 Jobs and grid applications 45
3.2 Application flow in a grid 45
3.2.1 Parallel flow 46
3.2.2 Serial flow 47
3.2.3 Networked flow 49
3.2.4 Jobs and sub-jobs 50
3.3 Job criteria 51
3.3.1 Batch job 51
3.3.2 Standard application 52
3.3.3 Parallel applications 52
3.3.4 Interactive jobs 53
3.4 Programming language considerations 53
3.5 Job dependencies on system environment 54
3.6 Checkpoint and restart capability 56
3.7 Job topology 56
3.8 Passing of data input/output 57
3.9 Transactions 58
3.10 Data criteria 58
3.11 Usability criteria 59
3.11.1 Traditional usability requirements 59
3.11.2 Usability requirements for grid solutions 59
3.12 Non-functional criteria 61
3.12.1 Software license considerations 62
3.12.2 Grid application development 66
3.13 Qualification scheme for grid applications 68
3.13.1 Knock-out criteria for grid applications 68
3.13.2 The grid application qualification scheme 69
3.14 Summary 69

Chapter 4. Data management considerations 71
4.1 Data criteria 73
4.1.1 Individual/separated data per job 73
4.1.2 Shared data access 74
4.1.3 Locking 75
4.1.4 Temporary data spaces 76
4.1.5 Size of data 76

4.1.6 Network bandwidth 77
4.1.7 Time-sensitive data 77
4.1.8 Data topology 77
4.1.9 Data types 79
4.1.10 Data volume and grid scalability 80
4.1.11 Encrypted data 84
4.2 Data management techniques and solutions 85
4.2.1 Shared file system 85
4.2.2 Databases 86
4.2.3 Replication (distribution of files across a set of nodes) 86
4.2.4 Mirroring 87
4.2.5 Caching 87
4.2.6 Transfer agent 88
4.2.7 Access Control System 88
4.2.8 Peer-to-peer data transfer 88
4.2.9 Sandboxing 89
4.2.10 Data brokering 90
4.2.11 Global file system approach 90
4.2.12 SAN approach 93
4.2.13 Distributed approach 95
4.2.14 Database solutions for grids 98
4.2.15 Data brokering 100
4.3 Some data grid projects in the Globus community 102
4.3.1 EU DataGrid 102
4.3.2 GriPhyn 102
4.3.3 Particle Physics Data Grid 103
4.4 Summary 103

Chapter 5. Getting started with development in C/C++ 105
5.1 Overview of programming environment 106
5.1.1 Globus libc APIs 106
5.1.2 Makefile 106
5.1.3 Globus module 109
5.1.4 Callbacks 109
5.2 Submitting a job 110
5.2.1 Shells commands 111
5.2.2 globusrun 113
5.2.3 GSIssh 114
5.2.4 Job submission skeleton for C/C++ applications 117
5.2.5 Simple broker 124
5.3 Summary 132

Chapter 6. Programming examples for Globus using Java 133

6.1 CoGs 134
6.2 GSI/Proxy 134
6.3 GRAM 138
6.3.1 GramJob 138
6.3.2 GramJobListener 138
6.3.3 GramException 139
6.4 MDS 140
6.4.1 Example of accessing MDS 141
6.5 RSL 145
6.5.1 Example using RSL 145
6.6 GridFTP 148
6.6.1 GridFTP basic third-party transfer 149
6.6.2 GridFTP client-server 151
6.6.3 URLCopy 154
6.7 GASS 155
6.7.1 Batch GASS example 156
6.7.2 Interactive GASS example 158
6.8 Summary 161

Chapter 7. Using Globus Toolkit for data management 163
7.1 Using a Globus Toolkit data grid with RSL 165
7.2 Globus Toolkit data grid low-level API: globus_io 169
7.2.1 globus_io example 172
7.2.2 Skeleton source code for creating a simple GSI socket 173
7.3 Global access to secondary storage 178
7.3.1 Easy file transfer by using globus_gass_copy API 178
7.3.2 globus_gass_transfer API 187
7.3.3 Using the globus_gass_server_ez API 188
7.3.4 Using the globus-gass-server command 192
7.3.5 Globus cache management 192
7.4 GridFTP 194
7.4.1 GridFTP examples 195
7.4.2 Globus GridFTP APIs 195
7.5 Replication 208
7.5.1 Shell commands 209
7.5.2 Replica example 209
7.5.3 Installation 211
7.6 Summary 213

Chapter 8. Developing a portal 215
8.1 Building a simple portal 216
8.2 Integrating portal function with a grid application 232
8.2.1 Add methods to execute the Globus commands 232

8.2.2 Putting it together 236
8.3 Summary 244

Chapter 9. Application examples 245
9.1 Lottery simulation program 246
9.1.1 Simulate a lottery using gsissh in a shell script 246
9.1.2 Simulate a lottery using Globus commands 254
9.2 Small Blue example 262
9.2.1 Gridification 265
9.2.2 Implementation 268
9.2.3 Compilation 275
9.2.4 Execution 276
9.3 Hello World example 278
9.3.1 The Hello World application 280
9.3.2 Dynamic libraries dependencies 281
9.3.3 Starting the application by the resource provider 285
9.3.4 Compilation 286
9.3.5 Execution 287
9.4 Summary 288

Chapter 10. Globus Toolkit V3.0 289
10.1 Overview of changes from GT2 to GT3 290
10.1.1 SOAP message security 290
10.1.2 Creating grid services 290
10.1.3 Security - proxies 291
10.1.4 SOAP GSI plugin for C/C++ Web services 291
10.2 OGSI implementation 291
10.3 Open Grid Service Architecture (OSGA) 292
10.4 Globus grid services 293
10.4.1 Index Services 293
10.4.2 Service data browser 293
10.4.3 GRAM 293
10.4.4 Reliable File Transfer Service (RFT) 296
10.4.5 Replica Location Service (RLS) 296
10.5 Summary 296

Appendix A. Grid qualification scheme 297
A suggested grid application qualification scheme 298

Appendix B. C/C++ source code for examples 305
Globus API C++ wrappers 306
ITSO_GASS_TRANSFER 306
ITSO_GLOBUS_FTP_CLIENT 311
ITSO_CB 315

ITSO_GRAM_JOB 316
StartGASSServer() and StopGASSServer() 324
ITSO broker 327
SmallBlue example 331
HelloWorld example 341
Lottery example 349
C/C++ simple examples 355
gassserver.C 355
Checking credentials 357
Submitting a job 358

Appendix C. Additional material 365
Locating the Web material 365
Using the Web material 366
How to use the Web material 366

Related publications 367
IBM Redbooks 367
Other publications 367
Online resources 369
How to get IBM Redbooks 372

Index 373

Figures

1-1 Possible user view of grid 7
1-2 Security in a grid environment 8
1-3 Broker service 8
1-4 Scheduler 9
1-5 Data management 10
1-6 GRAM 10
2-1 MDS overview 24
2-2 Standard file transfer 27
2-3 Third-party file transfer 27
2-4 Share job information for fault-tolerance 32
2-5 Grid portal on an application server 35
2-6 Grid topologies 39
3-1 Parallel application flow 47
3-2 Serial job flow 48
3-3 Networked job flow 49
3-4 Job with sub-jobs in a grid application 50
4-1 Federated DBMS architecture 75
4-2 Data topology of a grid 78
4-3 Independently working jobs on disjunct data subsets 81
4-4 Static input data processed by jobs with changing parameters 82
4-5 All jobs works on the same data and write on the same data set 83
4-6 Jobs with individual input data writing output into one data store 84
4-7 Sandboxing 89
4-8 Accessing Avaki Data Grid through NFS locally mounted file system . . 92
4-9 Avaki share mechanism 93
4-10 Storage Tank architecture 95
4-11 Replica logical view 96
4-12 File replication in a data grid between two organizations 98
4-13 Federated databases 99
5-1 GSI-enabled OpenSSH architecture 114
5-2 Job submission using non-blocking calls 118
5-3 Working with a broker 125
5-4 GQ LDAP browser 127
7-1 Data management interfaces 164
7-2 File staging 167
7-3 Using globus_io for secure communication 172
7-4 GASS Copy example 180
7-5 Replica example 210

© Copyright IBM Corp. 2003. All rights reserved.

8-1 Sample grid portal login screen 216
8-2 Simple grid portal welcome screen 217
8-3 Simple grid portal application flow 218
8-4 Simple grid portal login flow 219
8-5 Simple grid portal application submit flow 222
8-6 Simple grid portal application information and logout flow 230
9-1 Lottery example 247
9-2 Lottery example using Globus commands 255
9-3 Problem suitable for Grid enablement 263
9-4 Gridified SmallBlue 266
9-5 How to transfer an object via GRAM and GASS 267
9-6 Cluster model 279
9-7 Grid model 280
9-8 Hello World example with dynamic library dependencies issues 281
10-1 Globus Toolkit V3 job invocation 294

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

© Copyright IBM Corp. 2003. All rights reserved.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS®
AIX®
CICS®
DB2®
DB2 Connect™
DFS™

IBM®
ibm.com®
Lotus Notes®
Lotus®
Notes®
MQSeries®
Redbooks™
Redbooks (logo)
Sametime®
Storage Tank™
Tivoli®
WebSphere®
xSeries™

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook is the second in a planned series of redbooks addressing grid computing. In the first redbook, *Introduction to Grid Computing with Globus*, SG24-6895, grid concepts and the Globus Toolkit were introduced. In this redbook, we build on those concepts to discuss enabling applications to run in grid environments. Again, we focus on the open source Globus Toolkit.

In Chapters 1–4 of this publication, we look at various factors that should be considered when contemplating creating or porting an application to a grid environment. These factors include infrastructure considerations, application-specific considerations, and data management considerations. As a result, readers should come away with an appreciation for the applicability of a grid environment to their particular application(s).

In the latter part of the book, we provide detailed information and examples of using the Globus Toolkit to develop grid-enabled applications. We provide examples both in C/C++ and in Java.

This is not intended to be a complete programmers guide or reference manual. Instead we focus on many of the issues that an architect or developer needs to be aware of when designing a grid-enabled application. The programming samples provided in this publication provide the basic techniques required to get you started in the exciting world of application development for grid environments.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Bart Jacob is a Senior Technical Staff Member at IBM Corp - International Technical Support Organization, Austin Center. He has 23 years of experience providing technical support across a variety of IBM products and technologies, including communications, object-oriented software development, and systems management. He has over10 years of experience at the ITSO, where he has been writing IBM Redbooks™ and creating and teaching workshops around the world on a variety of topics. He holds a Masters degree in Numerical Analysis from Syracuse University.

© Copyright IBM Corp. 2003. All rights reserved.

Luis Ferreira also known as “Luix”, is a Software Engineer at IBM Corporation - International Technical Support Organization, Austin Center, working on Linux and grid computing projects. He has 19 years of experience with UNIX-like operating systems, and holds a MSc Degree in SystemS Engineering from Universidade Federal do Rio de Janeiro in Brazil. Before joining the ITSO, Luis worked at Tivoli® Systems as a Certified Tivoli Consultant, at IBM Brasil as a Certified IT Specialist, and at Cobra Computadores as a Kernel Developer and Software Designer.

Norbert Bieberstein is a Solution Development Manager at IBM Software Group in EMEA, located in Düsseldorf, Germany, working with system integrators and ISVs for about five years. He has also worked as an IT architect at IBM's application architecture project office in Somers, New York, and, before that, as a consultant for CASE and software engineering at IBM’s software development labs. This work resulted in a book on CASE technology that was published in Germany in 1993. He has organized several educational events for IBMers, business partners, and customers on IBM SW products. In 1997 he acted as the coordinating editor of the awarded IBM Systems Journal, 1/97 Edition. Before joining IBM in 1989, he worked as an application and system developer for a software vendor of CIM/ERP systems. Norbert holds a Masters degree in mathematics from the University of Technology Aachen, Germany, and developed their evaluation systems for electron microscopes. He is currently studying for an MBA at Henley Management School. He has written extensively on application architecture considerations.

Candice Gilzean is a Software Engineer for the Grid Computing Initiative in Austin, Texas. She has 2.5 years of experience for IBM Global Services. She holds a BS degree in Computer Science with a minor in Math from Texas Tech University. Her areas of expertise include Grid Computing, .Net programming, and AIX®. She has written extensively on job flow, java cog, and Globus ToolKit Version 3.

Jean-Yves Girard is an IT Specialist working in the Grid Design Center for e-business on demand, IBM Server Group, located in France in Montpellier. He has been with IBM for six years and has been involved with Grid Computing since March 2002 and Linux solutions for three years. He holds a degree of “élève ingénieur” from École Centrale Paris (France). His areas of expertise include Linux operating system, HPC computing, Grid technologies, and Web Services.

Roman Strachowski is an IT Specialist working in the Sales Operations Team, IBM Sales & Distribution Group, located in Zurich, Switzerland. He has been with IBM for five years and has been involved with Grid Computing since December 2002. Roman was working for two years in IBM ICAM as a system administrator. After that he went to the Swiss Lotus® Notes® development team and, 1 year

ago joined the Sales Operations team as a developer. He holds a degree of “Informatiker” from "GIBZ" in Switzerland. His areas of expertise include Java Development, Lotus Notes Development & Administration, Lotus Sametime® Development, Intelligent Agent Development, Linux operating system, Grid technologies, and Web Services.

Seong (Steve) Yu is an Advisory Software Engineer on the WebSphere® Beta team, IBM Software Group, in Austin,

Texas. He has been with IBM for 20 years, the last three years with the WebSphere product. Steve led the development of a WebSphere V5 migration guide redbook. He developed the original design and the prototype migration tool for JSP and Servlet migration. Steve has a BS degree in Mathematics/Computer Science from UCLA and an MS degree in Computer Science from CSU, and he is currently pursuing a PhD in Computer Science at NSU, Ft. Lauderdale. His current research interests include autonomic computing, machine learning, and multi-agent systems.

Thanks to the following people for their contributions to this project:

Paul Bate
IBM Global Services Architecture and Technology Center of Excellence

Andreas Hermelink
Grid Computing WW Technical Sales Enablement, IBM Somers

Rob High
WebSphere Chief Architect and Distinguished Engineer, IBM Austin

Susan Malaika
IBM Senior Technical Staff Member, IBM Silicon Valley Lab

Jean-Pierre Prost
Grid Expert at the IBM Grid Design Center for eBusiness on Demand, IBM Montpellier

Duane Quintern
IBM Global Services

Masanobu Takagi
Web Technologies, Technical Practice, Competency Management, IGS-Japan

Julie Czubik
International Technical Support Organization, Poughkeepsie Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com®/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an Internet note to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. JN9B Building 003 Internal Zip 2834
  11400 Burnet Road
  Austin, Texas 78758-3493

1

Introduction

Grid computing is gaining a lot of attention within the IT industry. Though it has been used within the academic and scientific community for some time, standards, enabling technologies, toolkits and products are becoming available that allow businesses to utilize and reap the advantages of grid computing.

As with many emerging technologies, you will find almost as many definitions of grid computing as the number of people you ask. However, one of the most used toolkits for creating and managing a grid environment is the Globus Toolkit. Therefore we will present most of our information and concepts within the context of the Globus Toolkit.

Note that though most of our examples and testing were accomplished in a Linux environment, grid computing in general implies support for multiple platforms and platform transparency.

It is recommended that the first publication in this series, *Introduction to Grid Computing with Globus*, SG24-6895, be used as a companion to this publication, as many of the concepts and details of the Globus Toolkit provided in the first book will not be duplicated here.

The first part of this book discusses considerations related to what kind of applications are good candidates for grid environments. This discussion is based on both grid characteristics and application characteristics. In later chapters, we provide examples of using the Globus Toolkit to enable applications to run in a grid environment. We also dedicate a large portion of this publication to

© Copyright IBM Corp. 2003. All rights reserved.

data-handling considerations. Though it is one thing to execute one or more application components on various nodes within a grid, one must also consider how the application will access and transfer data in an efficient and secure manner.

Important: Though we are focusing on the Globus Toolkit for this publication, it is important to note that there are other providers of solutions for grid computing. Some of these build on top of the Globus Toolkit to provide key services not directly addressed by Globus.

IBM has worked closely with these other vendors and sees their work as strategic to the overall evolution of grid solutions. Though the products and technologies that these vendors are developing are not necessarily addressed in this publication, they are important and should be considered when building a gird or enabling applications for a grid environment. Some of these vendors include:

- Platform Computing: Provides a set of solutions to help connect, manage, service, and optimize resources in a grid environment

  http://www.platform.com

- Avaki: Solutions to address data access and integration

  http://www.avaki.com

- DataSynapse: Supplies a CPU scavenging platform

  http://www.datasynapse.com

- United Devices: Provides a grid application framework for CPU scavenging

  http://www.ud.com

1.1 High-level overview of grid computing

The most common description of grid computing includes an analogy to a power grid. When you plug an appliance or other object requiring electrical power into a receptacle, you expect that there is power of the correct voltage to be available, but the actual source of that power is not known. Your local utility company provides the interface into a complex network of generators and power sources and provides you with (in most cases) an acceptable quality of service for your energy demands. Rather than each house or neighborhood having to obtain and maintain their own generator of electricity, the power grid infrastructure provides a virtual generator. The generator is highly reliable and adapts to the power needs of the consumers based on their demand.

The vision of grid computing is similar. Once the proper kind of infrastructure is in place, a user will have access to a virtual computer that is reliable and adaptable to the user's needs. This virtual computer will consist of many diverse computing resources, but these individual resources will not be visible to the user, just as the consumer of electric power is unaware of how his electricity is being generated.

To reach this vision, there must be standards for grid computing that will allow a secure and robust infrastructure to be built. Standards such as the Open Grid Services Architecture (OGSA) and tools such as those provided by the Globus Toolkit provide the necessary framework.

Initially, businesses will build their own infrastructures (what we might call intra-grids), but over time, these grids will become interconnected. This interconnection will be made possible by standards such as OGSA and the analogy of grid computing to the power grid will become real.

1.1.1 Types of grids

Grid computing can be used in a variety of ways to address various kinds of application requirements. Often, grids are categorized by the type of solutions that they best address. The three primary types of grids are summarized below. Of course, there are no hard boundaries between these grid types, and often grids may be a combination of two or more of these. But as you consider developing applications that may run in a grid environment, the type of grid environment that you will be using will affect many of your decisions.

Computational

A computational grid is focused on setting aside resources specifically for compute power. In this type of grid most of the machines are high-performance servers.

Scavenging

A scavenging grid is most commonly used with large numbers of desktop machines. Machines are scavenged for available CPU cycles and other resources. Owners of the desktop machines are usually given control of when their resources are available to participate in the grid.

Data grid

A data grid is responsible for housing and providing access to data across multiple organizations. Users are not concerned with where this data is located as long as they have access to the data. For example, you may have two universities doing life science research, each with unique data. A data grid would allow them to share their data, manage the data, and manage security issues such as who has access to what data.

Another common distributed computing model that is often associated with, or confused with, grid computing is peer-to-peer computing. In fact, some consider this another form of grid computing. For a more detailed analysis and comparison of grid computing and peer-to-peer computing, refer to *On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing,* by Ian Foster, Adriana Iamnitchi. This document can be found at:

`http://people.cs.uchicago.edu/~anda/papers/foster_grid_vs_p2p.pdf`

1.2 Globus Project

The Globus Project is a joint effort on the part of researchers and developers from around the world that are focused on the concept of grid computing. It is organized around four main activities:

- Research
- Software Tools
- Testbeds
- Applications

You can get more information about the Globus Project and their activities at:

`http://www.globus.org`

For this publication, we present most of our information in the context of the Globus Toolkit. The Globus Toolkit provides software tools to make it easier to build computational grids and grid-based applications. The Globus Toolkit is both an open architecture and open source toolkit.

At the time of this writing, the current version of the Globus Toolkit is Version 2.2. Version 3 of the toolkit is currently in alpha release and is expected to be

available shortly. However, the examples and information we provide are mostly based on the 2.2 version.

1.2.1 Globus Toolkit Version 2.2

The Globus Toolkit V2.2 provides:

- A set of basic facilities needed for grid computing:
  - Security: Single sign-on, authentication, authorization, and secure data transfer
  - Resource Management: Remote job submission and management
  - Data Management: Secure and robust data movement
  - Information Services: Directory services of available resources and their status.
- Application Programming Interfaces (APIs) to the above facilities
- C bindings (header files) needed to build and compile programs

In addition to the above, which are considered the core of the toolkit, other components are also available that complement or build on top of these facilities.

For instance, Globus provides a rapid development kit known as Commodity Grid (CoG), which supports technologies such as Java, Python, Web services, CORBA, and so on.

The facilities provided by Globus can be used to build grids and grid-enabled applications today. Many such environments have been built. However, when building such an infrastructure that is suitable for use in business environments, there are other considerations that have not been fully addressed by the Globus Toolkit V2.2. For instance, services such as life-cycle management, accounting and charge back systems, and other facilities may be desired or required. Another consideration when building a grid environment today is the ability to interconnect with other grids in the future. To enable the interconnection between grids developed by different organizations that may be using different technologies requires standards to be put in place and adopted.

1.2.2 OGSA and Globus Toolkit V3

The Open Grid Services Architecture (OGSA) is an evolving standard for which there is much industry support. Globus Toolkit Version 3 will be the reference implementation for OGSA.

OGSA addresses both issues we discussed in the previous section. First it changes the programming model to one that supports the concept of the various

facilities becoming available as Web services. This will provide multiple benefits, including:

- A common and open standards-based set of ways to access various grid services using standards such as SOAP, XML, and so on
- The ability to add and integrate additional services such as life cycle management in a seamless manner
- A standard way to find, identify, and utilize new grid services as they become available

In addition to benefits such as these, OGSA will provide for inter-operability between grids that may have been built using different underlying toolkits.

As mentioned, Globus Toolkit Version 3 will be the reference implementation for OGSA. Though the programming model will change, most of the actual APIs that are available with Globus Toolkit V2.2 will remain the same. Therefore, work done today to implement a grid environment and enable applications will not necessarily be lost.

OGSA and OGSI

OGSA defines a standard for the overall structure and services to be provided in grid environments. The Open Grid Services Interface (OGSI) specification is a companion standard that defines the interfaces and protocols that will be used between the various services in a grid environment. The OGSI is the standard that will provide the inter-operability between grids designed using OGSA.

1.3 Grid components: A high-level perspective

In this section we describe at a high level the primary components of a grid environment. Depending on the grid design and its expected use, some of these components may or may not be required, and in some cases they may be combined to form a hybrid component. However, understanding the roles of the components as we describe them here will help you understand the considerations for enabling applications as discussed throughout the rest of the book.

1.3.1 Portal - User interface

Just as a consumer sees the power grid as a receptacle in the wall, likewise a grid user should not see all of the complexities of the computing grid. Though the user interface could come in many forms and be application specific, for the purposes of our discussion let us think of it as a portal. Most users today

understand the concept of a Web portal, where their browser provides a single interface to access a wide variety of information sources.

A grid portal provides the interface for a user to launch applications that will utilize the resources and services provided by the grid. From this perspective the user sees the grid as a virtual computing resource just as the consumer of power sees the receptacle as an interface to a virtual generator.

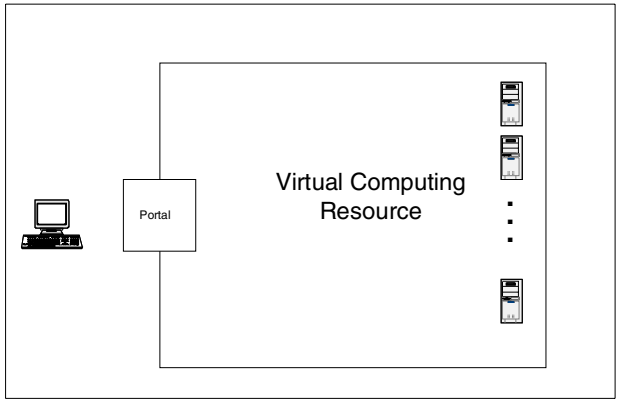


Figure 1-1 Possible user view of grid

The current Globus Toolkit does not provide any services or tools to generate a portal, but this can be accomplished with tools such as WebSphere.

1.3.2 Security

A major requirement for grid computing is security. At the base of any grid environment, there must be mechanisms to provide security including authentication, authorization, data encryption, and so on. The Grid Security Infrastructure (GSI) component of the Globus Toolkit provides robust security mechanisms. The GSI includes an OpenSSL implementation. It also provides a single sign-on mechanism, so once a user is authenticated, a proxy certificate is created and used when performing actions within the grid.

When designing your grid environment, you may use the GSI sign-in to grant access to the portal or you may have your own security for the portal. The portal would then be responsible for signing into the grid, either using the user's credentials, or using a generic set of credentials for all authorized users of the portal.

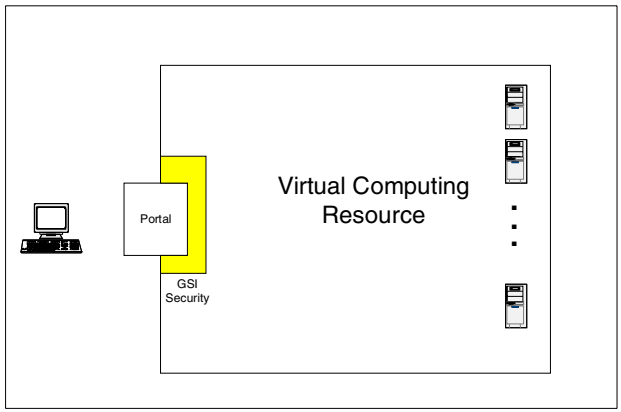


Figure 1-2 Security in a grid environment

1.3.3 Broker

Once authenticated, the user will be launching an application. Based on the application, and possibly on other parameters provided by the user, the next step is to identify the available and appropriate resources to utilize within the grid. This task could be carried out by a broker function. Though there is no broker implementation provided by Globus, there is an LDAP-based information service. This service is called Grid Information Service (GIS), or more commonly the Monitoring and Discovery Service (MDS). This service provides information about the available resources within the grid and their statuses. A broker service could be developed that utilizes MDS.

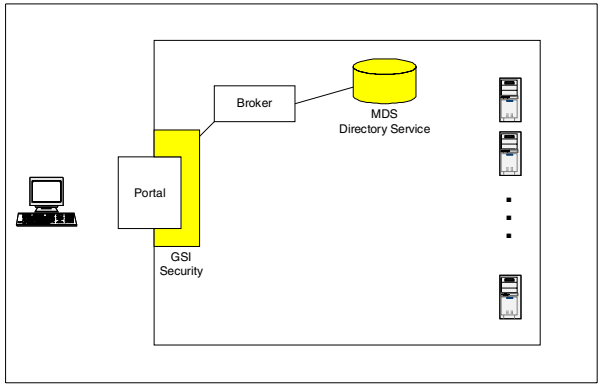


Figure 1-3 Broker service

1.3.4 Scheduler

Once the resources have been identified, the next logical step is to schedule the individual jobs to run on them. If a set of standalone jobs are to be executed with no interdependencies, then a specialized scheduler may not be required. However, if it is desired to reserve a specific resource or to ensure that different jobs within the application run concurrently (for instance, if they require inter-process communication), then a job scheduler should be used to coordinate the execution of the jobs.

The Globus Toolkit does not include such a scheduler, but there are several schedulers available that have been tested with and can be utilized in a Globus grid environment.

It should also be noted that there could be different levels of schedulers within a grid environment. For instance, a cluster could be represented as a single resource. The cluster may have its own scheduler to help manage the nodes it contains. A higher level scheduler (sometimes called a meta scheduler) might be used to schedule work to be done on a cluster, while the cluster's scheduler would handle the actual scheduling of work on the cluster's individual nodes.

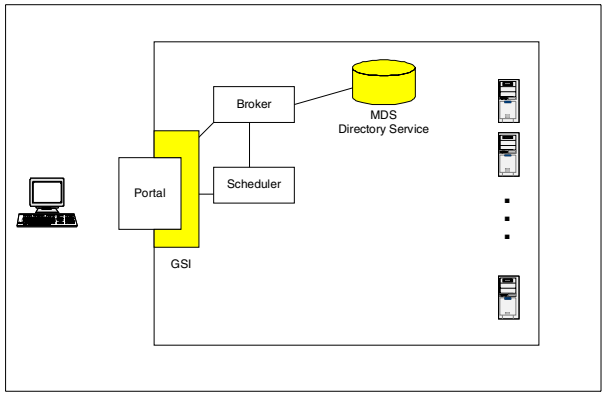


Figure 1-4 Scheduler

1.3.5 Data management

If any data (including application modules) must be moved or made accessible to the nodes where an application's jobs will execute, then there needs to be a secure and reliable method for moving files and data to various nodes within the grid. The Globus Toolkit contains a data management component that provides such services. This component, known as Grid Access to Secondary Storage (GASS), includes facilities such as GridFTP. GridFTP is built on top of the standard FTP protocol, but adds additional functions and utilizes the GSI for user

authentication and authorization. Therefore, once a user has an authenticated proxy certificate, she can utilize the GridFTP facility to move files without having to go through a login process to every node involved. This facility provides third party file transfer so that one node can initiate a file transfer between two other nodes.

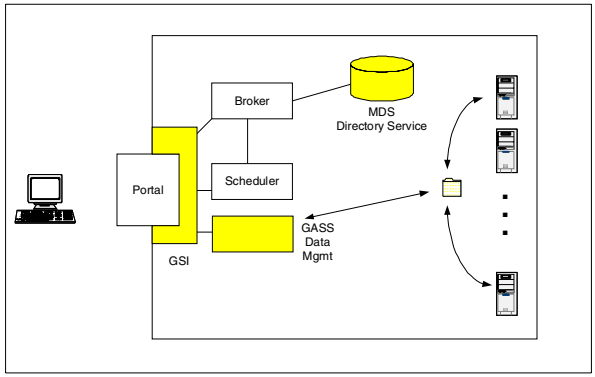


Figure 1-5 Data management

1.3.6 Job and resource management

With all of the other facilities we have just discussed in place, we now get to the core set of services that help perform actual work in a grid environment. The Grid Resource Allocation Manager (GRAM) provides the services to actually launch a job on a particular resource, check on its status, and retrieve its results when it is complete.

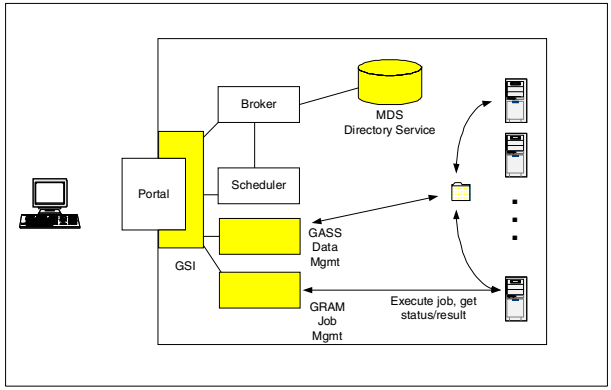


Figure 1-6 GRAM

1.3.7 Other

There are other facilities that may need to be included in your grid environment and considered when designing and implementing your application. For instance, inter-process communication and accounting/chargeback services are two common facilities that are often required. We will not discuss these in detail in this publication, but they are certainly important considerations.

1.4 Job flow in a grid environment

In the preceding section we provided a brief and high-level view of the primary components of a grid environment. As you start thinking about enabling an application for a grid environment, it is important to keep in mind these components and how they relate and interact with one another.

Depending on your grid implementation and application requirements, there are many ways in which these pieces can be put together to create a solution.

1.5 Summary

Grid computing is becoming a viable option in enterprises with the emergence and maturation of key technologies and open standards such as OGSA and OGSI.

In this chapter we have provided a high-level overview of the key facilities that make up grid environments. In the next chapter we will describe these in more detail and start describing the various considerations for enabling applications to take advantage of grid environments.

2

Grid infrastructure considerations

A grid computing environment provides the virtual computing resource that will be used to execute applications. As you are considering application design and implementation, it is important to understand the infrastructure that makes up this virtual computing environment.

When considering whether an application is a good candidate to execute in a grid environment, one must first understand the basic structure of a grid, the services that are and are not provided, and how this can affect the application. Once you understand these considerations, you will have a better idea of what facilities your application will need to use and how.

In the last chapter, we provided a high-level view of how a typical grid and the required services might be structured and how a job and its related data might flow through the grid.

In this chapter, we discuss various components and services in more detail and highlight specific issues and considerations that you should be aware of as you architect and develop your grid application.

© Copyright IBM Corp. 2003. All rights reserved.

2.1 Grid infrastructure components

This section describes the grid infrastructure components and how they map with the Globus Toolkit. It will also address how each of the components can affect the application architecture, design, and deployment.

The main components of a grid infrastructure are security, resource management, information services, and data management.

Security is an important consideration in grid computing. Each grid resource may have different security policies that need to be complied with. A single sign-on authentication method is a necessity. A commonly agreed upon method of negotiating authorization is also needed.

When a job is submitted, the grid resource manager is concerned with assigning a resource to the job, monitoring its status, and returning its results.

For the grid resource manager to make informed decisions on resource assignments, the grid resource manager needs to know what grid resources are available, and their capacities and current utilization. This knowledge about the grid resources is maintained and provided by Grid Information Service (GIS), also known as the Monitoring and Discovery Service (MDS).

Data management is concerned with how jobs transfer data or access shared storage.

Let us look at each of these components in more detail.

2.1.1 Security

Security is an important component in the grid computing environment. If you are a user running jobs on a remote system, you care that the remote system is secure to ensure that others do not gain access to your data.

If you are a resource provider that allows jobs to be executed on your systems, you must be confident that those jobs cannot corrupt, interfere with, or access other private data on your system.

Aside from these two perspectives, the grid environment is subject to any and all other security concerns that exist in distributed computing environments.

The Globus Toolkit, at its base, has the Grid Security Infrastructure (GSI), which provides many facilities to help manage the security requirements of the grid environment.

As you are developing applications targeted for a grid environment, you will want to keep security in mind and utilize the facilities provided by GSI.

The security functions within the grid architecture are responsible for the authentication, authorization, and secure communication between grid resources.

Grid security infrastructure (GSI)

Let us see how GSI provides authentication, authorization, and secure communications.

Authentication

GSI contains the infrastructure and facilities to provide a single sign-on environment. Through the **`grid-proxy-init`** command or its related APIs, a temporary proxy is created based on the user's private key. This proxy provides authentication and can be used to generate trusted sessions and allow a server to decide on the user's authorization.

A proxy must be created before a user can submit a job to be run or transfer data through the Globus Toolkit facilities. Depending on the configuration of the environment, a proxy may or may not be required to query the information services database.

Other facilities are available outside of the Globus Toolkit, such as GSI-Enabled OpenSSH, that utilize the same authentication mechanism to create secure communications channels.

For more information on the GSI-Enabled OpenSSH, visit:

`http://grid.ncsa.uiuc.edu/ssh/`

Authorization

Authentication is just one piece of the security puzzle. The next step is authorization. That is, once a user has been authenticated to the grid, what they are authorized to do.

In GSI this is handled by mapping the authenticated user to a local user on the system where a request has been received.

The proxy passed by the operation request (such as a request to run a job) contains a distinguished name of the authenticated user. A file on the receiving system is used to map that distinguished name to a local user.

Through this mechanism, either every user of the grid could have a user ID on each system within the grid (which would be difficult to administer if the number of systems in the grid becomes large and changes often), or users could be

assigned to virtual groups. For example, all authenticated users from a particular domain may be mapped to run under a common user ID on a particular resource. This helps separate the individual user ID administration for clients from the user administration that must be performed on the various resources that make up the grid.

Grid secure communication

It is important to understand the communication functions within the Globus Toolkit. By default, the underlying communication is based on the mutual authentication of digital certificates and SSL/TLS.

To allow secure communication within the grid, the OpenSSL package is installed as part of the Globus Toolkit. It is used to create an encrypted tunnel using SSL/TSL between grid clients and servers.

The digital certificates that have been installed on the grid computers provide the mutual authentication between the two parties. The SSL/TLS functions that OpenSSL provides will encrypt all data transferred between grid systems. These two functions together provide the basic security services of authentication and confidentiality.

Other grid communication

If you cannot physically access your grid client or server, it may be necessary to gain remote access to the grid. While your operating system's default telnet program works fine for remote access, the transmission of the data is in clear text. That means that the data transmission would be vulnerable to someone listening or sniffing the data on the network. While this vulnerability is low, it does exist and needs to be dealt with.

To secure the remote communication between a client and grid server, the use of Secure Shell (SSH) can be used. SSH will establish an encrypted session between your client and the grid server. Using a tool such as the GSI-Enabled OpenSSH, you get the benefits of the secure shell while also using the authentication mechanism already in place with GSI.

Application enablement considerations - Security

When designing grid-enabled applications, security concerns must be taken into consideration. The following list provides a summary of some of these considerations.

- Single sign-on: ID mapping across systems

  GSI provides the authentication, authorization, and secure communications as described above. However, the application designer needs to fully understand the security administration and implications. For instance:

  - Is it acceptable to have multiple users mapped to the same user ID on a target system?
  - Must special auditing be in place to understand who actually launched the application?
  - The application should be independent of the fact that different user ID mappings may be used across the different resources in the grid.

- Multi-platform

  Though the GSI is based on open and standardized software that will run on multiple platforms, the underlying security mechanisms of various platforms will not always be consistent. For instance, the security mechanisms for reading, writing, and execution on traditional Unix or Linux-based systems is different than for a Microsoft Windows environment. The application developer should take into account the possible platforms on which the application may execute.

- Utilize GSI

  For any application-specific function that might also require authentication or special authorization, the application should be designed to utilize GSI in order to simplify development and the user's experience by maintaining the single sign-on paradigm.

- Data encryption

  Though GSI, in conjunction with the data-management facilities covered later, provides secure communication and encryption of data across the network, the application designer should also take into account what happens to the data after it has arrived at its destination. For instance, if sensitive data is passed to a resource to be processed by a job and is written to the local disk in a non-encrypted format, other users or applications may have access to that data.

2.1.2 Resource management

The grid resource manager is concerned with resource assignments as jobs are submitted. It acts as an abstract interface to the heterogeneous resources of the

grid. The resource management component provides the facilities to allocate a job to a particular resource, provides a means to track the status of the job while it is running and its completion information, and provides the capability to cancel a job or otherwise manage it.

In Globus, the remote job submission is handled by the Globus Resource Allocation Manager (GRAM).

Globus Resource Allocation Manager (GRAM)

When a job is submitted by a client, the request is sent to the remote host and handled by a gatekeeper daemon. The gatekeeper creates a job manager to start and monitor the job. When the job is finished, the job manager sends the status information back to the client and terminates.

The GRAM subsystem consists of the following elements:

- The **`globusrun`** command and associated APIs
- Resource Specification Language (RSL)
- The gatekeeper daemon
- The job manager
- Dynamically-Updated Request Online Coallocator (DUROC)

Each of these elements are described briefly below.

The globusrun command

The **`globusrun`** command (or its equivalent API) submits a job to a resource within the grid. This command is typically passed an RSL string (see below) that specifies parameters and other properties required to successfully launch and run the job.

Resource Specification Language (RSL)

RSL is a language used by clients to specify the job to be run. All job submission requests are described in an RSL string that includes information such as the executable file; its parameters; information about redirection of stdin, stdout, and stderr; and so on. Basically it provides a standard way of specifying all of the information required to execute a job, independent of the target environment. It is then the responsibility of the job manager on the target system to parse the information and launch the job in the appropriate way.

The syntax of RSL is very straightforward. Each statement is enclosed within parenthesis. Comments are designated with parenthesis and asterisks, for example, (* this is a comment *). Supported attributes include the following:

- rsl_substitution: Defines variables
- executable: The script or command to be run

- arguments: Information or flags to be passed to the executable
- stdin: Specifies the remote URL and local file used for the executable
- stdout: Specifies the remote file to place standard output from the job
- stderr: Specifies the remote file to place standard error from the job
- queue: Specifies the queue to submit the job (requires a scheduler)
- count: Specifies the number of executions
- directory: Specifies the directory to run the job
- project: Specifies a project account for the job (requires a scheduler)
- dryRun: Verifies the RSL string but does not run the job
- maxMemory: Specifies the maximum amount of memory in MBs required for the job
- minMemory: Specifies the minimum amount of memory in MBs required for the job
- hostCount: Specifies the number of nodes in a cluster required for the job
- environment: Specifies environment variables that are required for the job
- jobType: Specifies the type of job single process, multi-process, mpi, or condor
- maxTime: Specifies the maximum execution wall or cpu time for one execution
- maxWallTime: Specifies the maximum walltime for one execution
- maxCpuTime: Specifies the maximum cpu time for one execution
- gramMyjob: Specifies the whether the gram myjob interface starts one process/thread (independent) or more (collective)

The following examples show how RSL scripts are used with the **globusrun** command. The following is a list of files included in this example:

- MyScript.sh: Shell script that executes the **ls -al** and **ps -ef** commands.

```
#!/bin/sh -x
ls -al
ps -ef
```

- MyTest.rsl: RSL script that calls the shell script /tmp/MySrcipt.sh. It runs the script in the /tmp directory and stores the standard output of the script in /tmp/temp. The contents are below.

```
& (rsl_substitution  = (TOPDIR  "/tmp"))(executable = $(TOPDIR)/MyScript.sh
) (directory=/tmp)(stdout=/tmp/temp)(count = 1)
```

- MyTest2.rsl: RSL script that executes the **/bin/ps -ef** command and stores the standard output of the script in /tmp/temp2.

```
& (rsl_substitution  = (EXECDIR "/bin"))(executable = $(EXECDIR)/ps )
(arguments=ef)(directory=/tmp)(stdout=/tmp/temp)(count = 1)
```

In Example 2-1, the **globusrun** command is used with MyTest.rsl to execute MyTest.sh on the resource (system) t3. The output of the script stored in /tmp/temp is then displayed using the Linux **more** command.

Example 2-1 Executing MyTest.sh with MyTest.rsl

```
[t3user@t3 guser]$ globusrun -r t3 -f MyTest.rsl
globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
[t3user@t3 guser]$ more /tmp/temp
total 116
drwxrwxrwt    9 root     root         4096 Mar 12 15:45 .
drwxr-xr-x   22 root     root         4096 Feb 26 20:44 ..
drwxrwxrwt    2 root     root         4096 Feb 26 20:45 .ICE-unix
-r--r--r--    1 root     root           11 Feb 26 20:45 .X0-lock
drwxrwxrwt    2 root     root         4096 Feb 26 20:45 .X11-unix
drwxrwxrwt    2 xfs      xfs          4096 Feb 26 20:45 .font-unix
-rw-r--r--    1 t3user   globus          0 Mar 10 11:57 17487_output
[t3user@t3 guser]$
```

In Example 2-2, MyTest2.rsl is used to display the currently executing processes using the **ps** command.

Example 2-2 Executing ps -ef with MyTest.2.rsl

```
[t3user@t3 guser]$ globusrun -r t3 -f MyTest2.rsl
globus_gram_client_callback_allow successful
GRAM Job submission successful
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE
[t3user@t3 guser]$ more /tmp/temp2
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Feb26 ?        00:00:04 init
root         2     1  0 Feb26 ?        00:00:00 [keventd]
root         2     1  0 Feb26 ?        00:00:00 [keventd]
root         3     1  0 Feb26 ?        00:00:00 [kapmd]
root         4     1  0 Feb26 ?        00:00:00 [ksoftirqd_CPU0]
root         5     1  0 Feb26 ?        00:00:09 [kswapd]
root         6     1  0 Feb26 ?        00:00:00 [bdflush]
root         7     1  0 Feb26 ?        00:00:01 [kupdated]
root         8     1  0 Feb26 ?        00:00:00 [mdrecoveryd]
```

```
root        12     1  0 Feb26 ?        00:00:20 [kjournald]
root        91     1  0 Feb26 ?        00:00:00 [khubd]
root       196     1  0 Feb26 ?        00:00:00 [kjournald]
[t3user@t3 guser]$
```

Although there is no way to directly run RSL scripts with the **globus-job-run** command, the command utilizes RSL to execute jobs. By using the dumprsl parameter, **globus-job-run** is a useful tool to build and understand RSL scripts.

Example 2-3 Using globus-job-run -dumprsl to generate RSL

```
[t3user@t3 guser]$ globus-job-run -dumprsl t3 /tmp/MyScript
  &(executable="/tmp/MyTest")
[t3user@t3 guesr]$
```

Gatekeeper

The gatekeeper daemon provides a secure communication mechanism between clients and servers. The gatekeeper daemon is similar to the inetd daemon in terms of functionality. However, the gatekeeper utilizes the security infrastructure (GSI) to authenticate the user before launching the job. After authentication, the gatekeeper initiates a job manager to launch the actual job and delegates the authority to communicate with the client.

Job manager

The job manager is created by the gatekeeper daemon as part of the job requesting process. It provides the interfaces that control the allocation of each local resource. It may in turn utilize other services such as job schedulers. The default implementation performs the following functions and forks a new process to launch the job:

- Parses the RSL string passed by the client
- Allocates job requests to local resource managers
- Sends callbacks to clients, if necessary
- Receives status requests and cancel requests from clients
- Sends output results to clients using GASS, if requested

Dynamically-Updated Request Online Coallocator (DUROC)

The Dynamically-Updated Request Online Coallocator (DUROC) API allows users to submit multiple jobs to multiple GRAMs with one command. DUROC uses a coallocator to execute and manage these jobs over several resource managers. To utilize the DUROC API you can use RSL (described above), the API within a C program, or the **globus-duroc** command.

The RSL script that contains the DUROC syntax is parsed at the GRAM client and allocated to different job managers.

Application enablement considerations - Resource Mgmt

There are several considerations for application architecture, design, and deployment related to resource management.

In its simplest form GRAM is used by issuing a **globusrun** command to launch a job on a specific system. However, in conjunction with MDS (usually through a broker function), the application must ensure that the appropriate target resource(s) are used. Some of the items to consider include:

- Choosing the appropriate resource

  By working in conjunction with the broker, ensure that an appropriate target resource is selected. This requires that the application accurately specifies the required environment (operating system, processor, speed, memory, and so on). The more the application developer can do to eliminate specific dependencies, the better the chance that an available resource can be found and that the job will complete.

- Multiple sub-jobs

  If an application includes multiple jobs, the designer must understand (and maybe reduce) their interdependencies. Otherwise, they will have to build logic to handle items such as:

  – Inter-process communication
  – Sharing of data
  – Concurrent job submissions

- Accessing job results

  If a job returns a simple status or a small amount of output, the application may be able to simply retrieve the data from stdout and stderr. However, the capturing of that output will need to be correctly specified in the RSL string that is passed to the **globusrun** command. If more complex results must be retrieved, the GASS facility may need to be used by the application to transfer data files.

- Job management

  GRAM provides mechanisms to query the status of the job as well as perform operations such as cancelling the job. The application may need to utilize these capabilities to provide feedback to the user or to clean up or free up resources when required. For instance, if one job within an application fails, other jobs that may be dependent on it may need to be cancelled before needlessly consuming resources that could be used by other jobs.

2.1.3 Information services

Information services is a vital component of the grid infrastructure. It maintains knowledge about resource availability, capacity, and current utilization. Within

any grid, both CPU and data resources will fluctuate, depending on their availability to process and share data. As resources become free within the grid, they can update their status within the grid information services. The client, broker, and/or grid resource manager uses this information to make informed decisions on resource assignments.

The information service is designed to provide:

- Efficient delivery of state information from a single source
- Common discovery and enquiry mechanisms across all grid entities

Information service providers are programs that provide information to the directory about the state of resources. Examples of information that is gathered includes:

- Static host information

  Operating system name and version, processor vendor/model/version/ speed/cache size, number of processors, total physical memory, total virtual memory, devices, service type/protocol/port

- Dynamic host information

  Load average, queue entries, and so on

- Storage system information

  Total disk space, free disk space, and so on

- Network information

  Network bandwidth, latency, measured and predicted

- Highly dynamic information

  Free physical memory, free virtual memory, free number of processors, and so on

The Grid Information Service (GIS), also known as the Monitoring and Discovery Service (MDS), provides the information services in Globus. The MDS uses the Lightweight Directory Access Protocol (LDAP) as an interface to the resource information.

Monitoring and Discovery Service (MDS)

MDS provides access to static and dynamic information of resources. Basically, it contains the following components:

- Grid Resource Information Service (GRIS)
- Grid Index Information Service (GIIS)
- Information providers
- MDS client

Figure 2-1 represents a conceptual view of the MDS components. As illustrated, the resource information is obtained by the information provider and it is passed to GRIS. GRIS registers its local information with the GIIS, which can optionally also register with another GIIS, and so on. MDS clients can query the resource information directly from GRIS (for local resources) and/or a GIIS (for grid-wide resources).

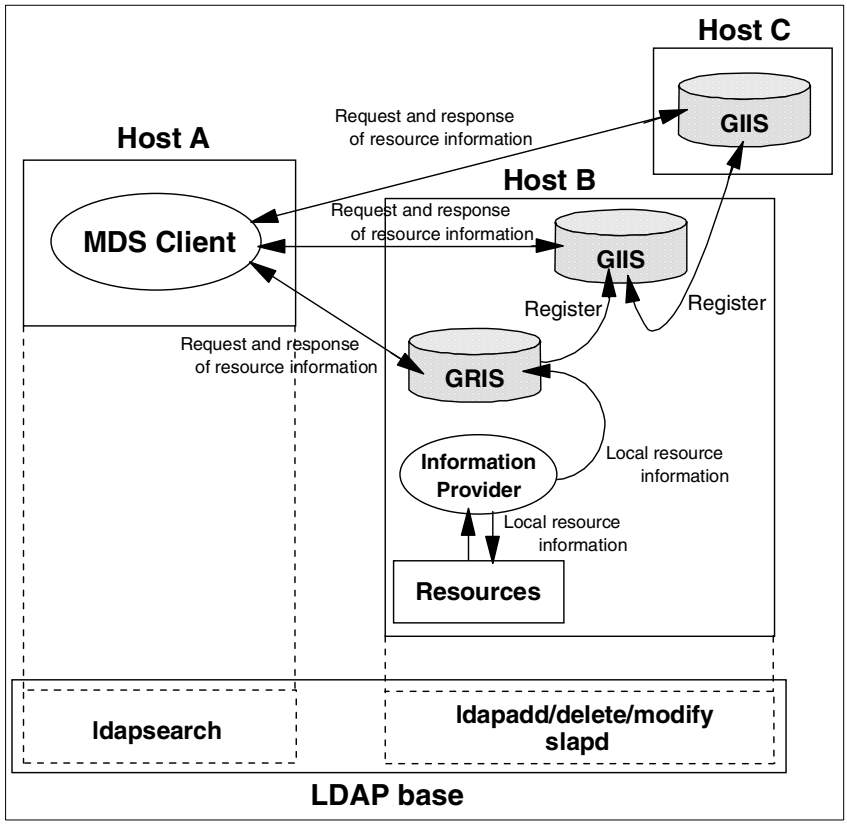


Figure 2-1 MDS overview

Grid Resource Information Service (GRIS)

GRIS is the repository of local resource information derived from information providers. GRIS is able to register its information with a GIIS, but GRIS itself does not receive registration requests. The local information maintained by GRIS is updated when requested, and cached for a period of time known as the time-to-live (TTL). If no request for the information is received by GRIS, the information will time out and be deleted. If a later request for the information is received, GRIS will call the relevant information provider(s) to retrieve the latest information.

Grid Index Information Service (GIIS)

GIIS is the repository that contains indexes of resource information registered by the GRIS and other GIISs. It can be seen as a grid-wide information server. GIIS has a hierarchical mechanism, like DNS, and each GIIS has its own name. This means client users can specify the name of a GIIS node to search for information.

Information providers

The information providers translate the properties and status of local resources to the format defined in the schema and configuration files. In order to add your own resource to be used by MDS, you must create specific information providers to transfer the properties and status to GRIS.

MDS client

The MDS client is based on the LDAP client command, `ldapsearch`, or an equivalent API. A search for information about resources in the grid environment is initially performed by the MDS client.

Application enablement considerations - Information Services

Considerations related to information services include:

- It is important to fully understand the requirements for a specific job so that the MDS query can be correctly formatted to return resources that are appropriate.
- Ensure that the proper information is in MDS. There is a large amount of data about the resources within the grid that is available by default within the MDS. However, if your application requires special resources or information that is not there by default, you may need to write your own information providers and add the appropriate fields to the schema. This may allow your application or broker to query for the existence of the particular resource/requirement.
- MDS can be accessed anonymously or through a GSI authenticated proxy. Application developers will need to ensure that they pass an authenticated proxy if required.
- Your grid environment may have multiple levels of GIIS. Depending on the complexity of the environment and its topology, you want to ensure that you are accessing an appropriate GIIS to search for the resources you require.

2.1.4 Data management

When building a grid, the most important asset within your grid is your data. Within your design, you will have to determine your data requirements and how you will move data around your infrastructure or otherwise access the required data in a secure and efficient manner. Standardizing on a set of grid protocols

will allow you to communicate between any data source that is available within your design.

You also have choices for building a federated database to create a virtual data store or other options including Storage Area Networks, network file systems, and dedicated storage servers.

Globus provides the GridFTP and Global Access to Secondary Storage (GASS) data transfer utilities in the grid environment. In addition, a replica management capability is provided to help manage and access replicas of a data set. These facilities are briefly described below.

GridFTP

The GridFTP facility provides secure and reliable data transfer between grid hosts. Its protocol extends the File Transfer Protocol (FTP) to provide additional features including:

- Grid Security Infrastructure (GSI) and Kerberos support allows for both types of authentication. The user can set various levels of data integrity and/or confidentiality.
- Third-party data transfer allows a third party to transfer files between two servers.
- Parallel data transfer using multiple TCP streams to improve the aggregate bandwidth. It supports the normal file transfer between a client and a server. It also supports the third-party data transfers between two servers.
- Striped data transfer that partitions data across multiple servers to further improve aggregate bandwidth.
- Partial file transfer that allows the transfer of a portion of a file.
- Reliable data transfer that includes fault recovery methods for handling transient network failures, server outages, and so on. The FTP standard includes basic features for restarting failed transfer. The GridFTP protocol exploits these features, and substantially extends them.
- Manual control of TCP buffer size allows achieving maximum bandwidth with TCP/IP. The protocol also has support for automatic buffer size tuning.
- Integrated instrumentation. The protocol calls for restart and performance markers to be sent back.

GridFTP server and client

Globus Toolkit provides the GridFTP server and GridFTP client, which are implemented by the in.ftpd daemon and by the **`globus-url-copy`** command (and related APIs), respectively. They support most of the features defined for the GridFTP protocol.

The GridFTP server and client support two types of file transfer: Standard and third party. The standard file transfer is where a client sends or retrieves a file to/from the remote machine, which runs the FTP server. An overview is shown in Figure 2-3.

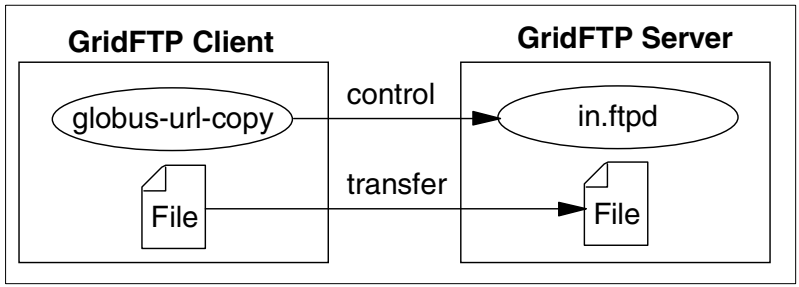


Figure 2-2 Standard file transfer

Third-party data transfer allows a third party to transfer files between two servers.

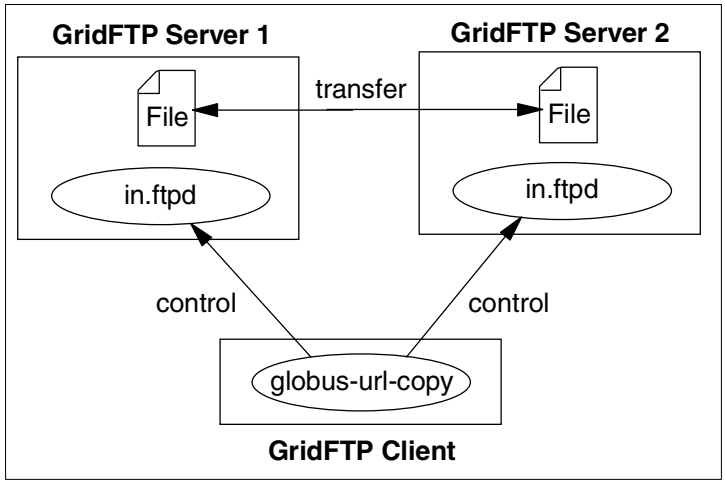


Figure 2-3 Third-party file transfer

Global Access to Secondary Storage (GASS)

GASS is used to transfer files between the GRAM client and the GRAM server. GASS also provides libraries and utilities for the opening, closing, and pre-fetching of data from datasets in the Globus environment. A cache management API is also provided. It eliminates the need to manually log into sites, transfer files, and install a distributed file system.

For further information, refer to the Globus GASS Web site:

`http://www-fp.globus.org/gass/`

Replica management

Another Globus facility for helping with data management is replica management. In certain cases, especially with very large data sets, it makes sense to maintain multiple replicas of all or portions of a data set that must be accessed by multiple grid jobs. With replica management, you can store copies of the most relevant portions of a data set on local storage for faster access. Replica management is the process of keeping track of where portions of the data set can be found.

Globus Replica Management integrates the Globus Replica Catalog (for keeping track of replicated files) and GridFTP (for moving data), and provides replica management capabilities for grids.

Application enablement considerations - Data management

Data management is concerned with collectively maximizing the use of the limited storage space, networking bandwidth, and computing resources. The following are some of the data management issues that need to be considered in application design and implementation:

- Dataset size

  For large datasets, it is not practical and may be impossible to move the data to the system where the job will actually run. Using data replication or otherwise copying a subset of the entire dataset to the target system may provide a solution.

- Geographically distributed users, data, computing and storage resources

  If your target grid is geographically distributed with limited network connection speeds, you must take into account design considerations around slow or limited data access.

- Data transfer over wide-area networks

  Take into account the security, reliability, and performance issues when moving data across the Internet or another WAN. Build the required logic to handle situations when the data access may be slow or prevented.

- Scheduling of data transfers

  There are at least two issues to consider here. One is the scheduling of data transfers so that the data is at the appropriate location at the time that it is needed. For instance, if a data transfer is going to take one hour and the data is required by a job that must run at 2:00AM, then schedule the data transfer in advance so that it is available by the time the job requires it.

  You should also be aware of the number and size of any concurrent file transfers to or from any one resource at the same time.

- Data replica selection

  If you are using the Globus Data Replication service, you will want to add the logic to your application to handle selecting the appropriate replica, that is, one that will contain the data that you need, while also providing the performance requirements that you have.

2.1.5 Scheduler

The Globus Toolkit does not provide a job scheduler or meta-scheduler. However, there are a number of job schedulers available that already are or can be integrated with Globus. For instance, the Condor-G product utilizes the Globus Toolkit and provides a scheduler designed for a grid environment.

Scheduling jobs and load balancing are important functions in the Grid.

Most grid systems include some sort of job-scheduling software. This software locates a machine on which to run a grid job that has been submitted by a user. In the simplest cases, it may just blindly assign jobs in a round-robin fashion to the next machine matching the resource requirements. However, there are advantages to using a more advanced scheduler.

Some schedulers implement a job-priority system. This is sometimes done by using several job queues, each with a different priority. As grid machines become available to execute jobs, the jobs are taken from the highest priority queues first. Policies of various kinds are also implemented using schedulers. Policies can include various kinds of constraints on jobs, users, and resources. For example, there may be a policy that restricts grid jobs from executing at certain times of the day.

Schedulers usually react to the immediate grid load. They use measurement information about the current utilization of machines to determine which ones are not busy before submitting a job. Schedulers can be organized in a hierarchy. For example, a meta-scheduler may submit a job to a cluster scheduler or other lower-level scheduler rather than to an individual machine.

More advanced schedulers will monitor the progress of scheduled jobs managing the overall work-flow. If the jobs are lost due to system or network outages, a good scheduler will automatically resubmit the job elsewhere. However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs should not be rescheduled. Typically, jobs have different kinds of completion codes, some of which are suitable for resubmission and some of which are not.

Reserving resources on the grid in advance is accomplished with a *reservation system*. It is more than a scheduler. It is first a calendar-based system for reserving resources for specific time periods and preventing any others from

reserving the same resource at the same time. It also must be able to remove or suspend jobs that may be running on any machine or resource when the reservation period is reached.

Condor-G

The Condor software consists of two parts, a resource-management part and a job-management part. The resource-management part keeps track of machine availability for running the jobs and tries to best utilize them. The job-management part submits new jobs to the system or put jobs on hold, keeps track of the jobs, and provides information about the job queue and completed jobs.

The machine with the resource-management part is referred to as the execution machine. The machine with the job-submission part installed is referred to as the submit machine. Each machine may have one or both parts. Condor-G provides the job management part of Condor. It uses the Globus Toolkit to start the jobs on the remote machine instead of the Condor protocols.

The benefits of using Condor-G include the ability to submit many jobs at the same time into a queue and to monitor the life-cycle of the submitted jobs with a built-in user interface. Condor-G provides notification of job completions and failures, and maintains the Globus credentials that may expire during the job execution. In addition, Condor-G is fault tolerant. The jobs submitted to Condor-G and the information about them are kept in persistent storage to allow the submission machine to be rebooted without losing the job or the job information. Condor-G provides exactly-once-execution semantics. Condor-G detects and intelligently handles cases such as the remote grid resource crashing.

Condor makes use of Globus infrastructure components such as authentication, remote program execution, and data transfer to utilize the grid resources. By using the Globus protocols, the Condor system can access resources at multiple remote sites. Condor-G uses the GRAM protocol for job submission and local GASS servers for file transfer.

Application enablement considerations - Scheduler

When considering enabling an application for a grid environment, there are several considerations related to scheduling. Some of these considerations include:

- Data management: Ensuring data is available when the job is scheduled to run. If data needs to be moved to the execution node, then data movement may also need to be scheduled.
- Communication: Any inter-process communication between related jobs will require that the jobs are scheduled to run concurrently.

- Scheduler's domain: In an environment with multiple schedulers, such as those with meta schedulers, the complexities of coordinating concurrent jobs, or ensuring certain jobs execute at a specific time, can become complex, especially if there are different schedulers for different domains.
- Scheduling policy: Scheduling can be implemented with different orientations:
  - Application oriented: Scheduling is optimized for best turn around time.
  - System oriented: Optimized for maximum throughput. A job may not be started immediately. It may be interrupted or preempted during execution. It may be scheduled to run overnight.
- Grid information service: The interaction between the scheduler and the information service can be complex. For instance, if the resource is found through MDS before the job is actually scheduled, then there may be an assumption that the current resource status will not change before execution of the job. Or a more proactive mechanism could be used to predict possible changes in the resource status so proactive scheduling decisions may be made.
- Resource broker: Typically a resource broker must interface with the scheduler.

2.1.6 Load balancing

Load balancing is concerned with the distribution of workload among the grid resources in the system. Though the Globus Toolkit does not provide a load-balancing function, under certain environments it is a desired service.

As the work is submitted to a grid job manager, the workload may be distributed in a push model, pull model, or combined model. A simple implementation of a push model could be built where the work is sent to grid resources in a round-robin fashion. However, this model does not consider the job queue lengths. If each grid resource is sent the same number of jobs, a long job queue could build up in some slower machines or a long-running job could block others from starting if not carefully monitored. One solution may be to use a weighted round-robin scheme.

In the pull model, the grid resources take the jobs from a job queue. In this model, synchronization and serialization of the job queue will be necessary to coordinate the taking of jobs by multiple grid resources. Local and global job queue strategies are also possible. In the local pull model strategy, each group of grid resources is assigned to take jobs from a local job queue. In the global pull model strategy, all the grid resources are assigned the same job queue. The advantage of the local pull model is the ability to partition the grid resources. For example, proximity to data, related jobs, or jobs of certain types requiring similar resources may be controlled in this way.

A combination of the push and the pull models may remove some previous concerns. The individual grid resources may decide when more work can be taken, and send a request for work to a grid job server. New work is then sent by the job server.

Failover conditions need to be considered in both of the load-balancing models. The non-operational grid resources need to be detected, and no new work should be sent to failed resources in the push model. In addition, all the submitted jobs that did not complete need to be taken care of in both push and pull models. All the uncompleted jobs in the failed host need to be either redistributed or taken over by other operational hosts in the group. This may be accomplished in one of two ways. In the simplest, the uncompleted jobs can be resent to another operational grid resource in the push model, or simply added back to the job queue in the pull model. In a more sophisticated approach, multiple grid resources may share job information such as the jobs in the queue and checkpoint information related to running jobs, as shown in Figure 2-4. In both models, the operational grid resources can take over the uncompleted jobs of a failed grid resource.

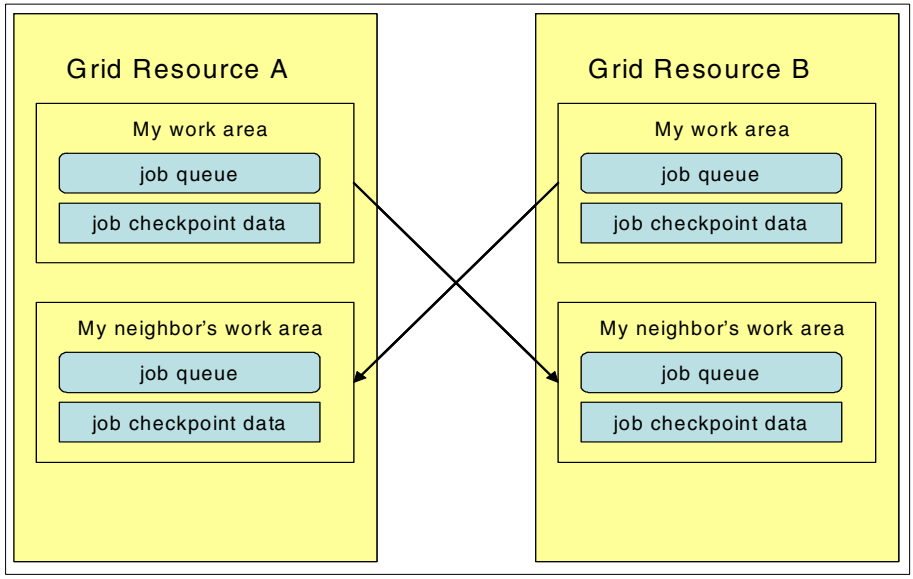


Figure 2-4 Share job information for fault-tolerance

Application-enablement considerations - Load balancing

When enabling applications for a grid environment, design issues related to load balancing may need to be considered. Based on the load-balancing mechanism that is in place (manual, push, pull, or some hybrid combination), the application designer/developer needs to understand how this will affect the application, and

specifically its performance and turn-around time. Applications with many individual jobs that each may be affected or controlled by a load-balancing system can benefit from the improved overall performance and throughput of the grid, but may also require more complicated mechanisms to handle the complexity of having its jobs delayed or moved to accommodate the overall grid.

2.1.7 Broker

As already described, the role of a broker in a grid environment can be very important. It is a component that will likely need to be implemented in most grid environments, though the implementation can vary from relatively simple to very complex.

The basic role of a broker is to provide match-making services between a service requester and a service provider. In the grid environment, the service requesters will be the applications or the jobs submitted for execution, and the service providers will be the grid resources.

With the advent of OGSA, the future service requester may be able to make requests of a grid service or a Web service via a generic service broker. A candidate for such a generic service broker may be IBM WebSphere Business Connection, which is currently a Web services broker.

The Globus toolkit does not provide the broker function. It does, however, provide the grid information services function through the Monitoring and Discovery Service (MDS). The MDS may be queried to discover the properties of the machines, computers, and networks such as the number of processors available at this moment, what bandwidth is provided, and the type of storage available.

Application enablement considerations - Broker

When designing an application for execution in a grid environment, it is important to understand how resources will be discovered and allocated. It may be up to the application to identify its resource requirements to the broker so that the broker can ensure that the proper and appropriate resources are allocated to the application.

2.1.8 Inter-process communications (IPC)

A grid system may include software to help jobs communicate with each other. For example, an application may split itself into a large number of sub-jobs. Each of these sub-jobs is a separate job in the grid. However, the application may implement an algorithm that requires that the sub-jobs communicate some information among them. The sub-jobs need to be able to locate other specific

sub-jobs, establish a communications connection with them, and send the appropriate data. The open standard Message Passing Interface (MPI) and any of several variations are often included as part of the grid system for just this kind of communication.

MPICH-G2

MPICH-G2 is an implementation of MPI optimized for running on grids. It combines easy secure job startup, excellent performance, data conversion, and multi-protocol communication. However, when communicating over wide-area networks, applications may encounter network congestion that severely impacts the performance of the application.

Application-enablement considerations - IPC

There are many possible solutions for inter-process communication, of which MPICH-G2 described above is just one. However, requiring inter-process communication between jobs always increases the complexity of an application, and when possible should be kept to a minimum. However, in large complex applications, it often cannot be avoided. In these cases, understanding the IPC mechanisms that are available and minimizing the effect of failed or slowed communications can help ensure the overall success of the applications.

2.1.9 Portal

A grid portal may be constructed as a Web page interface to provide easy access to grid applications. The Web user interface provides user authentication, job submission, job monitoring, and results of the job.

The Web user interface and interaction of a grid portal may be provided using an application server such as the WebSphere Application Server. See Figure 2-5 on page 35.

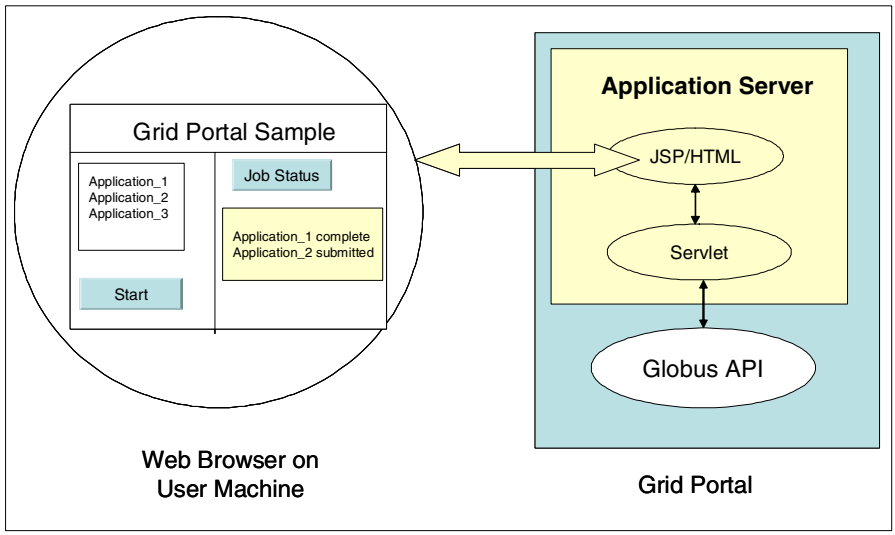


Figure 2-5 Grid portal on an application server

Application-enablement considerations - Portal

Whatever the user interface might be to your grid application, ease-of-use and the requirements of the user must be taken into account. As with any user interface, there are trade-offs between ease-of-use and the ability for advanced users to provide additional input to the application or to specify run-time parameters unique for a specific invocation of the job. By utilizing the GRAM facilities in the Globus Toolkit, it is also possible to obtain job status and to allow for job management such as cancelling a job in progress. When designing the portal, the users requirements in these areas must be understood and addressed.

Developing a portal for grid applications is described in more detail in Chapter 8, “Developing a portal” on page 215.

2.2 Non-functional requirements

The following sections describe some additional considerations related to the infrastructure. These considerations come under the heading of non-functional as they do not relate to a specific functional unit of the grid, such as job management, broker, and so on.

2.2.1 Performance

When considering enabling an application to execute in a grid environment, the performance of the grid and the performance requirements of the application must be considered. The service requester is interested in a quality of service that includes acceptable turnaround time. Of course, if building a grid and one or more applications that will be provided as a service on the grid, then the service provider also has interest in maximizing the utilization and throughput of the systems within the grid. The performance objectives of these two perspectives are discussed below.

Resource provider's perspective

The performance objective for a grid infrastructure is to achieve maximum utilization of the various resources within the grid to achieve maximum throughput. The resources may include but are not limited to CPU cycles, memory, disk space, federated databases, or application processing. Workload balancing and preemptive scheduling may be used to achieve the performance objectives. Applications may be allowed to take advantage of multiple resources by dividing the grid into smaller instances to have the work distributed throughout the grid. The goal is to take advantage of the grid as a whole to improve the application performance. The workload management can make sure that all resources within the grid are actively servicing jobs or requests within the grid.

Service requester's perspective

The turnaround time of an application running on the grid could vary depending on the type of grid resource used and the resource provider's quality-of-service agreement. For example, a quick turnaround may be achieved by submitting a processing-intensive standalone batch job to a high-performance grid resource. This assumes that the job is started immediately and that it is not preempted by another job during execution. The same batch job may be scheduled to run overnight when the resource demands are lower if a quick turnaround is not required. The resource provider may charge different prices for these two types of service.

If the application has many independent sub-jobs that can be scheduled for parallel execution, the turnaround time could be improved appreciably by running each sub-job on multiple grid hosts.

Turnaround time factors

This section discusses some of the factors that can impact the turnaround time of applications run on the grid resources.

Communication delays

Network speed and network latency can have significant impact to the application performance if it requires communicating with another application running on a remote machine. It is important to consider the proximity of the communicating applications to one another and the network speed and latency.

Data access delays

The network bandwidth and speed will be the critical factors for applications that need to access remote data. Proximity of the application to the data and the network capacity/speed will be important considerations.

Lack of optimization of the application to the grid resource

Optimum application performance is usually achieved by proper tuning and optimization on a particular operating system and hardware configuration. This poses possible issues if an application is simply loaded on a new grid host and run. This issue may be resolved if the service provider makes an arrangement with the resource provider so that the application's optimum configuration and resource requirements are identified ahead of time and applied when the application is run.

Contention for resource

Resource contention is always a problem when resources are shared. If resource contention impacts performance significantly, alternate resources may need to be introduced. For example, if a database is the source of contention, then introducing a replica may be an answer. In addition, the network may need to be divided to separate the traffic to the databases. Optimum sharing of the grid hosts may be achieved by a proper scheduling algorithm and workload balancing. For example, the shortest job first (SJF) batch job scheduling algorithm may provide the best turnaround time.

Reliability

Failures in the grid resource and network can cause unforeseen delays. To provide reliable job execution, the grid resource may apply various recovery methods for different failures. For example, in the checkpoint-restart environment, some amount of delay will be incurred each time a checkpoint is taken. A much longer delay may be experienced if the server crashed and the application was migrated to a new server to complete the run. In other instances, the delay may take the entire time to recover from a failure such as network outages.

2.2.2 Reliability

Reliability is always an issue in computing, and the grid environment is no exception. The best method of approaching this difficult issue is to anticipate all

possible failures and provide a means to handle them. The best reliability is to be surprise tolerant. The grid computing infrastructure must deal with host interruptions and network interruptions. Below are some approaches to dealing with such interruptions.

Checkpoint-restart

While a job is running, checkpoint images are taken at regular intervals. A checkpoint contains a snapshot of the job states. If a machine crashes or fails during the job execution, the job can be restarted on a new machine using the most recent checkpoint image. In this way, a long-running job that runs for months or even years can continue to run even though computers fail occasionally.

Persistent storage

The relevant state of each submitted job is stored in persistent storage by a grid manager to protect against local machine failure. When the local machine is restarted after a failure, the stored job information is retrieved. The connection to the job manager is reestablished.

Heartbeat monitoring

In a healthy heartbeat, a probing message is sent to a process and the process responds. If the process fails to respond, an alternate process may be probed. The alternate process can help to determine the status of the first process, and even restart it. However, if the alternate process also fails to respond then we assume that either the host machine has crashed or the network has failed. In this case, the client must wait until the communication can be reestablished.

System management

Any design will require a basic set of systems management tools to help determine availability and performance within the grid. A design without these tools is limited in how much support and information can be given about the health of the grid infrastructure. Alternate networks within a grid architecture can be dedicated to perform these functions so as to not hamper the performance of the grid.

2.2.3 Topology considerations

The distributed nature of grid computing makes spanning across geographies and organizations inevitable. As an intra-grid topology is extended to an inter-grid topology, the complexity increases. For example, the non-functional and operational requirements such as security, directory services, reliability, and performance become more complex. These considerations are discussed briefly in the following sections.

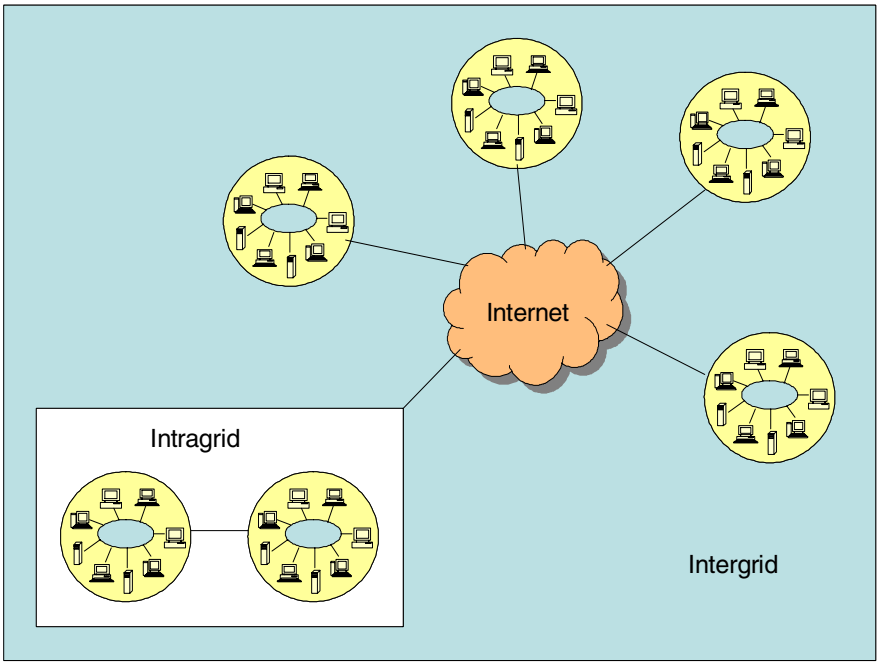


Figure 2-6 Grid topologies

Network topology

The network topology within the grid architecture can take on many different shapes. The networking components can represent the LAN or campus connectivity, or even WAN communication between the grid networks. The network's responsibility is to provide adequate bandwidth for any of the grid systems. Like many other components within the infrastructure, the networking can be customized to provide higher levels of availability, performance, or security.

Grid systems are for the most part network intensive due to security and other architectural limitations. For data grids in particular, which may have storage resources spread across the enterprise network, an infrastructure that is designed to handle a significant network load is critical to ensuring adequate performance.

The application-enablement considerations should include strategies to minimize network communication and to minimize the network latency. Assuming the application has been designed with minimal network communication, there are a number of ways to minimize the network latency. For example, a gigabit Ethernet

LAN could be used to support high-speed clustering or utilize high-speed Internet backbone between remote networks.

Data topology

It would be desirable to assign executing jobs to machines nearest to the data that these jobs require. This would reduce network traffic and possibly reduce scalability limits.

Data requires storage space. The storage possibilities are endless within a grid design. The storage needs to be secured, backed up, managed, and/or replicated. Within a grid design, you want to make sure that your data is always available to the resources that need it. Besides availability, you want to make sure that your data is properly secured, as you would not want unauthorized access to sensitive data. Lastly, you want more than decent performance for access to your data. Obviously, some of this relies on the bandwidth and distance to the data, but you will not want any I/O problems to slow down your grid applications. For applications that are more disk intensive, or for a data grid, more emphasis can be placed on storage resources, such as those providing higher capacity, redundancy, or fault-tolerance.

2.2.4 Mixed platform environments

A grid environment is a collection of heterogeneous hosts with various operating systems and software stacks. To execute an application, the grid infrastructure needs to know the application's prerequisites to find the matching grid host environment. Below are some things that the grid infrastructure must be aware of to ensure that applications can execute properly. It is equally as important for the application developer to consider these factors in order to maximize the kinds and numbers of environments on which the application will be able to execute.

Runtime considerations

The application's runtime requirements and the grid host's runtime environments must match. As an example, below are some considerations for Java applications. Similar requirements may exist for applications developed in other applications.

Java Virtual Machine (JVM)

Applications written in the Java programming language require the Java Virtual Machine (JVM). Java applications may be sensitive to the JVM version. To address this sensitivity, the application needs to identify the JVM version as a prerequisite. The prerequisite may specify the required JVM version or the minimum JVM version.

Java applications may be sensitive to the Java heap size. The Java application needs to specify the minimum heap size as part of its prerequisite.

Java packages such as J2SE or J2EE may also need to be identified as part of the prerequisites.

Availability of application across platforms (portability)

The executables of certain applications are platform specific. For example, an application written in the C or C++ programming language needs to be recompiled on the target platform before it can be run. The application could be pre-compiled for each platform and the resulting executables marked for a target platform. This will increase the number of qualifying grid host environments where the application can run. The limitation of this method will be the cost-effectiveness of porting the application to another platform.

Awareness of OS environment

The grid is a collection of heterogeneous computing resources. If the application has certain dependencies or requirements specific to the operating system, the application needs to verify that the correct environment is available and handle issues related to the differing environments.

Output file formats

The knowledge of the output file format is necessary when the output of an application running on one grid host is accessed by another application running on a different grid host. The two grid hosts may have different platform environments. XML may be considered as the data exchange format. XML has now become popular not only as a markup language for data exchange, but also as a data format for semi-structured data.

2.3 Summary

The functional components of a grid environment, as well as non-functional considerations such as performance requirements or operating system requirements, must be well understood when considering enabling an application to execute in a grid environment. This chapter has touched on many of these considerations.

In the next chapter, we look at the properties of an application itself to determine whether it is a good candidate to be grid enabled.

3

Application architecture considerations

In the previous chapters we have introduced grid computing, the Globus Toolkit and its components, and some of the considerations that the infrastructure can impose on a grid-enabled application.

In this chapter, we look at the characteristics of applications themselves. We provide guidance for deciding whether a particular application is well suited to run on a grid.

Often we find people assuming that for an application to gain advantage from a grid environment, it must be highly parallel or otherwise able to take advantage of parallel processing. In fact, some like to think of a grid as a distributed cluster. Although such parallel applications certainly can take advantage of a grid, you should not dismiss the use of grids for other types of applications as well. As introduced in Chapter 1, “Introduction” on page 1, a grid can be thought of as a virtual computing resource. Even a single threaded batch job could benefit from a grid environment by being able to run on any of a set of systems in the grid, taking advantage of unused cycles. A grid environment that can be used to execute any of a variety of jobs across multiple resources, transparently to the user, provides greater availability, reliability, and cost efficiencies than may exist with dedicated servers.

© Copyright IBM Corp. 2003. All rights reserved.

Similarly the need for large amounts of data storage can also benefit from a grid. Examples include thoroughly analyzing credit data for fraud detection, bankruptcy warning mechanisms, or usage patterns of credit cards. Operations on vast amounts of data by uniform calculations, such as the search for identifiable sequences in the human genome database, are also well suited for grid environments.

At some point, the question usually arises as to whether a problem should be solved in a grid or whether other approaches like HPC, Storage Tanks, and so on are sufficient. In order to decide on the best choice there are a number of aspects to consider from various perspectives. This chapter provides some basic ideas for dealing with the types of jobs and data in a grid.

This chapter also provides an overview of criteria that helps determine whether a given application qualifies for a grid solution. These criteria are discussed in four sections dealing with job/application, data, and usability and non-functional perspectives. Together they allow a sufficient understanding of the complexity, scope, and size of the grid application under consideration. It also allows the project team to detect any show stoppers and to size the effort and requirements needed to build the solution.

3.1 Jobs and grid applications

In order to have a clearer understanding of the upcoming discussion, we introduce the following terminology:

Grid Application A collection of work items to solve a certain problem or to achieve desired results using a grid infrastructure. For example, a grid application can be the simulation of business scenarios, like stock market development, that require a large amount of data as well as a high demand for computing resources in order to calculate and handle the large number of variables and their effects. For each set of parameters a complex calculation can be executed. The simulation of a large scale scenario then consists of a larger number of such steps. In other words, a grid application may consist of a number of jobs that together fulfill the whole task.

Job Considered as a single unit of work within a grid application. It is typically submitted for execution on the grid, has defined input and output data, and execution requirements in order to complete its task. A single job can launch one or many processes on a specified node. It can perform complex calculations on large amounts of data or might be relatively simple in nature.

3.2 Application flow in a grid

In this section, we look at the overall flow of a grid-enabled application, which may consist of multiple jobs. Traditional applications execute in a well known and somewhat static environment with fixed assets. We need to look at the considerations (and value) for having an application run in a grid environment where resources are dynamically allocated based on actual needs.

If taking advantage of multiple resources concurrently in a grid, you must consider whether the processing of the data can happen in parallel tasks or whether it must be serialized and the consequences of one job waiting for input data from another job. What may result is a network of processes that comprise the application.

Application flow vs job flow

For the remainder of the book an *application flow* is the flow of work between the jobs that make up the grid application. The internal flow of work within a job itself is called *job flow*.

Analyzing the type of flow within an application delivers the first determining factor of suitability for a grid. This does not mean that a complex networked application flow excludes implementation on a grid, nor does a simple flow type determine an easy deployment on a grid. Rather, besides the flow types, the sum of all qualifying factors allows for a good evaluation of how to enable an application for a grid.

There are three basic types of application flow that can be identified:

- Parallel
- Serial
- Networked

The following sections discuss each of these in more detail.

3.2.1 Parallel flow

If an application consists of several jobs that can all be executed in parallel, a grid may be very suitable for effective execution on dedicated nodes, especially in the case when there is no or a very limited exchange of data among the jobs.

From an initial job a number of jobs are launched to execute on preselected or dynamically assigned nodes within the grid. Each job may receive a discrete set of data, and fulfills its computational task independently and delivers its output. The output is collected by a final job or stored in a defined data store. Grid services, such as a broker and/or scheduler, may be used to launch each job at the best time and place within the grid.

Data producer and consumer

Jobs that produce output data are called *producers*, and jobs receiving input data are called *consumers*. Instead of an active job as the final consumer of data, there can be a defined data sink of any kind within the grid application. This could be a database record, a data file, or a message queue that consumes the data.

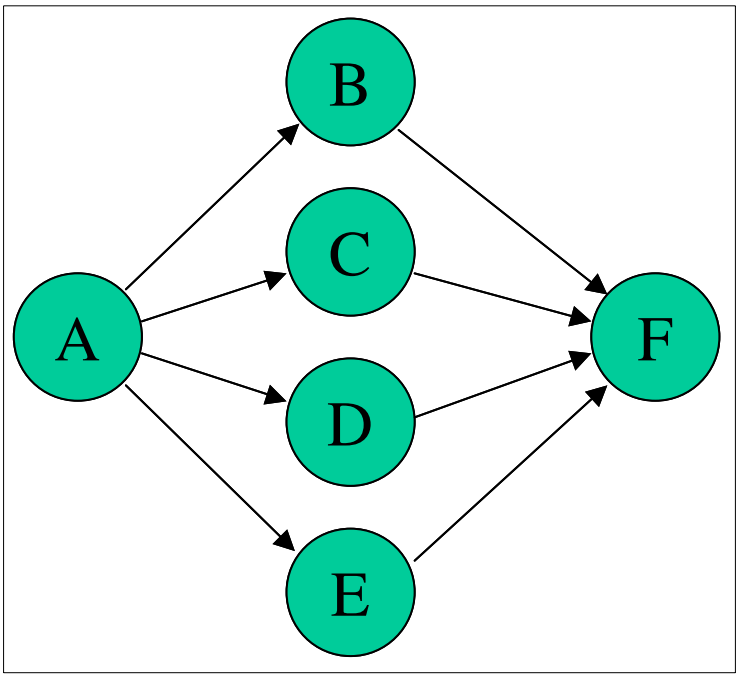


Figure 3-1 Parallel application flow

For a given problem or application it would be necessary to break it down into independent units. To take advantage of parallel execution in a grid, it is important to analyze tasks within an application to determine whether they can be broken down into individual and atomic units of work that can be run as individual jobs.

This parallel application flow type is well suited for deployment on a grid. Significantly, this type of flow can occur when there are separate data sets per job and none of the jobs need results from another job as input. For example, in the case of a simulation application that is based on a large array of parameter sets against which a specific algorithm is to be executed, a grid can help to deliver results more quickly. A larger coverage of the data sphere is reached when the jobs can run in parallel on as many suitable nodes as possible. Such a job can be as complex as a sophisticated spreadsheet script or any multidimensional mathematical formula of which each requires intense computing.

3.2.2 Serial flow

In contrast to the parallel flow is the serial application flow. In this case there is a single thread of job execution where each of the subsequent jobs has to wait for

its predecessor to end and deliver output data as input to the next job. This means any job is a consumer of its predecessor, the data producer.

In this case, the advantages of running in a grid environment are not based on access to multiple systems in parallel, but rather on the ability to use any of several appropriate and available resources. Note that each job does not necessarily have to run on the same resource, so if a particular job requires specialized resources, that can be accommodated, while the other jobs may run on more standard and inexpensive resources. The ability for the jobs to run on any of a number of resources also increases the application's availability and reliability. In addition, it may make the application inherently scalable by being able to utilize larger and faster resources at any particular point in time. Nevertheless when encountering such a situation it may be worthwhile to check whether the single jobs are really dependent of each other, or whether due to its nature they can be split into parallel executable units for submission on a grid.

Parallelization

Section 2.1 of *Introduction to Grid Computing with Globus*, SG24-6895, provides certain thoughts about parallelization of jobs for grids. For example, when dealing with mathematical calculations the commutative and associative laws can be exploited.

In iterative scenarios (for example, convergent approximation calculations) where the output of one job is required as input to the next job of the same kind, a serial job flow is required to reach the desired result. For best performance these kinds of processes might be executed on a single CPU or cluster, though performance is not always the primary criteria. Cost and other factors must also be considered, and once a grid environment is constructed such a job may be more cost effective when run on a grid versus utilizing a dedicated cluster.

An application may consist of a large number of such calculations where the start parameters are taken from a discrete set of values. Each resulting serial application flow then could be launched in parallel on a grid in order to utilize more resources. The serial flow A through D in Figure 3-2 is then replicated to A' through D', A" through D", and so forth.

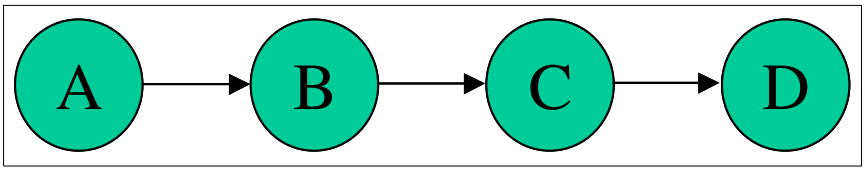


Figure 3-2 Serial job flow

In case it is not possible to completely convert a serial application flow into a parallel one, a networked application flow may result.

3.2.3 Networked flow

In this case (perhaps the most common situation), complexity comes into play.

As shown in Figure 3-3, certain jobs within the application are executable in parallel, but there are interdependences between them. In the example, jobs B and C can be launched simultaneously, but they heavily exchange data with each other. Job F cannot be launched before B and C have completed, whereas job E or D can be launched upon completion of B or C respectively. Finally, job G finally collects all output from the jobs D, E, and F, and its termination and results then represent the completion of the grid application.

Loose coupling

For a grid, this means the need for a job flow management service to handle the synchronization of the individual results. Loose coupling between the jobs avoids high inter-process communication and reduces overhead in the grid.

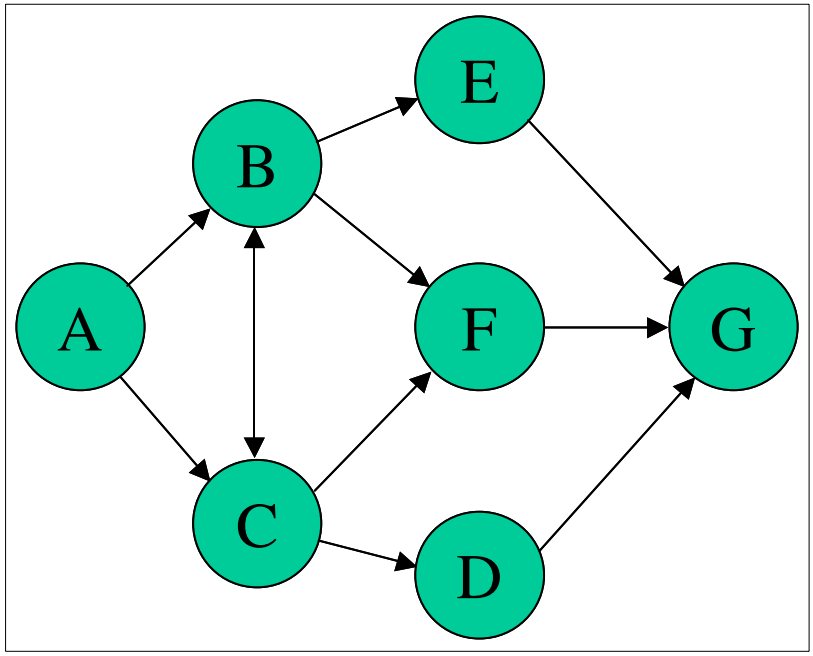


Figure 3-3 Networked job flow

For such an application you will need to do more analysis to determine how best to split the application into individual jobs, maximizing parallelism. It also adds more dependencies on the grid infrastructure services such as schedulers and brokers, but once that infrastructure is in place, the application can benefit from the flexibility and utilization of the virtualized computing environment.

3.2.4 Jobs and sub-jobs

Another approach to ease the managing of jobs within a grid application is to introduce a hierarchical system of sub-jobs. A job could utilize the services of the grid environment to launch one or more sub-jobs. For this kind of environment an application would be partitioned and designed in such a way that the higher-level jobs could include the logic to obtain resources and launch sub-jobs in whatever way is most optimal for the task at hand. This may provide some benefits for very large applications to isolate and pass the control and management of certain tasks to the individual components.

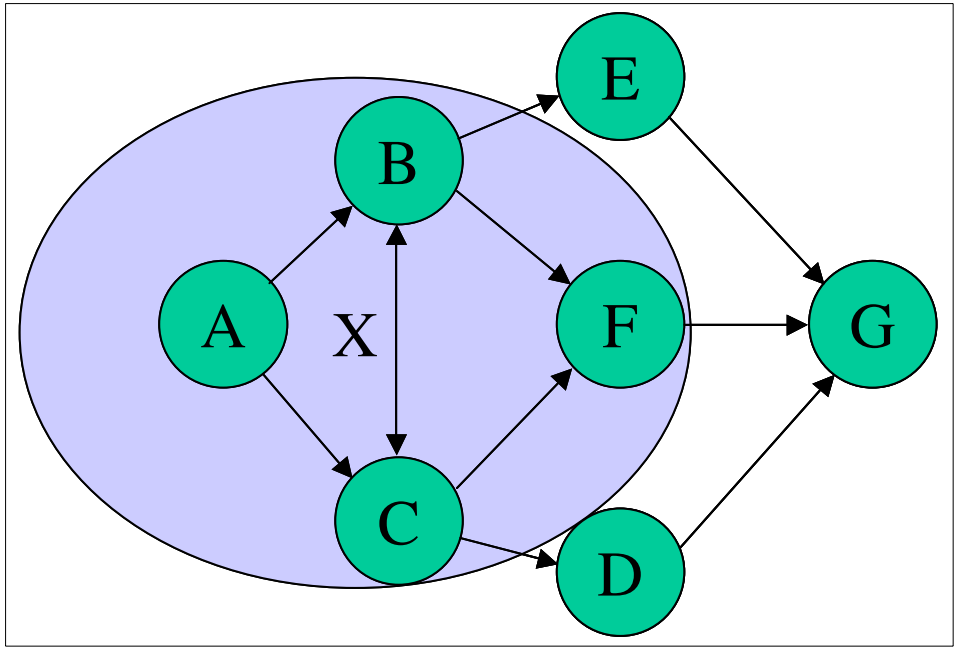


Figure 3-4 Job with sub-jobs in a grid application

As illustrated in Figure 3-4, in the shaded area named X, job A launches sub-jobs B and C, which communicate with each other and will launch another sub-job, F.

For the grid application, everything within the shaded area X may be regarded as one job, identified by job A. In this case the grid server or grid portal has to be

notified of either completion of the whole task under X in order to launch D and E respectively, or an explicit communication is established to handle notifications about partial completion of the tasks identified within job A by its sub-jobs B and C in order to run jobs E and D, respectively, on their own schedules.

In the latter case the grid services can take advantage of the available resources and gain the freedom to more efficiently distribute the workload. On the other hand, it generates more management requirements for the grid services, which can mean additional overhead. The grid architecture has to balance the individual advantages under the framework given by the available infrastructure and business needs.

3.3 Job criteria

A grid application consists of a number of jobs, which may often be executed in parallel. In this section the special requirements for each of these jobs are discussed.

A job as part of a grid application can theoretically be of any type: Batch, standard application, parallel application, and/or interactive.

3.3.1 Batch job

Jobs in a grid environment could be a traditional batch job on a mainframe or a program invoked via a command line interface in a Windows, Unix, or Linux environment. Normally, arguments are passed to the program, which can represent the data to process and parameter settings related to the job's execution.

Depending on its size and the network capacities, a batch job can be sent to the node along with its arguments and remotely launched for execution. The job can be a script for execution in a defined environment (for example, REXX, Java, or Perl script), or an executable program that has few or no special requirements for operating system versions, special DLLs to be linked to, JAR files to be in place or any other special environmental conditions.

The client, portal, and/or broker may need to know the specific requirements for the job so that the appropriate resource can be allocated.

The data for its computation are either transmitted as arguments or accessible by the job, be it in local or remote storage or in a file that can also be sent across the grid.

A batch job, especially one with few environmental requirements, in general is well suited for deployment in a grid environment.

3.3.2 Standard application

A grid environment can also be applicable to a standard application, like spreadsheets or video rendering systems. For example, if extensive financial calculations on many variations of similar input parameters are to be done, these could be processed on one or more nodes within the grid. See the Excel Grid example in Section 12.1 and “Zetagrid” in Section 11.4 of *Introduction to Grid Computing with Globus*, SG24-6895.

Often such a standard application requires an installation procedure and cannot be sent over the network to run simply as a batch job. However, a command line interface provided can be remotely used on a grid for execution of the application where it is installed.

In this case, the grid broker or grid portal needs to know the locations of the application and the availability of the node. The locations of the applications on the grid are relatively fixed, meaning in order to change it a new installation has to be performed and the application may need to be registered with the grid portal or grid server before it can be used.

New installations are mostly done manually as the applications often require certain OS conditions and application settings, or very often when installing on Windows a reboot needs to be executed. This makes a standard application in many cases quite difficult to handle on a grid, but does not exclude them. As advances in autonomic computing provide for self-provisioning, there will be less restrictions in this area.

Using standard software as jobs within a grid could raise licensing issues, either due to the desire to have the application installed on many different nodes in the grid, or related to single-user versus multi-user license agreements. We discuss more on licenses in a grid environment in 3.12.1, “Software license considerations” on page 62.

3.3.3 Parallel applications

Applications that already have a parallel application flow, such as those that have been designed to run in a cluster environment, may already be suited to run in a grid environment. In order to allow a grid server or grid portal to take the most advantage of these, there needs to be identifiable and accessible handles to the inner functions/jobs of such a parallel application. If this is not the case, such an application can only be handled as one unit, similar to a standard application. However, it makes sense to include such an application in a grid if the overall

task requires more than the resources available in a given cluster. This means that the grid could include several clusters with copies of a parallel application.

3.3.4 Interactive jobs

Interaction with a grid application is most commonly done via the grid portal or grid server interface. This implies that other than launching the job, there should not be on-going interaction between the user and the job.

Of course, if we go back to our initial view of the gird as a virtual computing resource, it is certainly feasible to think of an application requiring user interaction to be launched on any appropriate resource within the grid as long as a secure and reliable communications channel could be created and maintained between the user and the resource. Though the GSI-Enabled SSH package is available and could be used to create a secure session, the Globus Toolkit does not provide any tools or guidance for supporting such an application.

There would be many considerations and issues involved in the development and deployment of such an application within a grid environment. We will not discuss this type of application within the grid context any further in this publication.

3.4 Programming language considerations

Whenever an application is being developed, the question of the programming language to be used arises. The grid environment may include additional considerations.

Jobs that are made for high-performance computing are normally written in languages such as C or Fortran. Those jobs whose individual execution time does not play the most important role for the application, but whose contents and tasks are of more importance, may be written in other languages such as Java, or in scripting languages such as Perl.

Within a single grid application one might even consider writing various parts in different languages depending on the requirements for the individual jobs and available resources.

Some of the key considerations include:

- Portability to a variety of platforms

  This includes binary compatibility where languages such as Java provide an advantage, as a single binary can be executed on any platform supporting the

Java Virtual Machine. Interpreted languages such as Perl also tend to be portable, allowing the application to run no matter what the target platform.

Portability of source code can also be considered. For instance, one may decide to develop an application using C, and then compile it multiple times for a variety of target platforms. This will require additional work by the infrastructure to ensure that appropriate executables are distributed to any target resource.

- Run-time libraries/modules

  Depending on the language and how the program is linked, there may be a requirement for run-time libraries or other modules to be available. Again, the successful running of an application will depend on these libraries being available on, or moved to, the target resource.

- Interfaces to the grid infrastructure

  If the job must interface with the grid infrastructure, such as the Globus Toolkit, then the choice of language will depend on available bindings. For example, Globus Toolkit V2.2 includes bindings for C. However, through the CoG initiative, there are also APIs and bindings for Java, Perl and other languages. Note that an application may not have to interface with the Globus Toolkit directly, as it is more the responsibility of the infrastructure that is put in place. That is, given an appropriate infrastructure, the application may be developed such that it is independent of the grid-specific services.

One of the driving factors behind the OGSA initiative is to standardize on the way that various services and components of the grid infrastructure interface with one another. This provides programming language transparency between two communicating programs.That is, a program written in C, for example, could communicate with or through a service that is written in another language.

3.5 Job dependencies on system environment

As shown earlier, a grid application does not require a homogenous runtime environment, but there are certain considerations to be made in order to plan for the most beneficial deployment of it.

For any job in a grid application the following environmental factors may affect its operation. When developing an application, one must consider these factors and either design it to be as independent of these factors as possible, or understand that any dependencies will need to be taken into account within the grid infrastructure.

- *Operating Systems* version, service level, and OS parameter settings that are necessary for execution of the job, as well its reliance on certain system

services and auxiliary programs such as a registry. It is worthwhile to consider whether the grid application will be capable of running its jobs on any node with different operating systems or whether it will be restricted to a single operating system.

- *Memory size* required by a job may limit the possible nodes on which it can run. The available memory size depends not only on its physical presence at a node, but also on how much the operating system is capable of granting at run-time.
- *DLLs* that are to be linked for the execution of the job. They either need to be available on the target resource or could possibly be transferred and made available on the resource before the job is executed.
- *Compiler settings* play a role as compiler flags and locations may be different. For example, subtle differences like bit ordering, and number of bytes used for real and integer numbers may cause failures when a job is compiled on a different node or operating system than the one it will eventually be executed on.
- *Runtime environment* that has to be in place and ready to receive the job for execution. For instance, the right JDK or interpreter versions may have to be planned and in place.
- *Application Server* version and standard as well as its capacity may be needed to be considered as well as access requirements and services to be used.
- *Other applications* that are needed to properly run a job have to be in place prior to deployment of the grid application. These applications can be compilers, databases, system services such as the registry under Windows, and so on.
- *Hardware devices* that are required for certain jobs to perform their tasks. For example, requirements for storage, measurement devices, and other peripherals must be considered when building the application and planning the grid architecture.

When developing the grid application, these prerequisites need to be checked in order to avoid too many restrictions for job execution. A large number of restrictions could mean more complicated enablement as well as limiting the number of possible nodes on which the job will be able to run. Therefore, it is better to restrict such requirements during development of the application such that jobs can run in as generic an environment as possible.

3.6 Checkpoint and restart capability

A job within a grid application may be designed to be launched, perform its tasks, and report back to the user or grid portal regarding its success or failure. In the latter case the same job may be launched for a second time, if it has not changed any persistent data prior to its error state. This process can be then repeated until final successful completion. However, it may make sense that failures be handled by the grid server to allow a more sophisticated way to get to job completion.

By building checkpoint and restart capabilities into the job and making its state available to other services within the grid, the job could be restarted where it failed, even on a different node.

3.7 Job topology

For a grid application, there are various topology-related considerations. There are certain architectural requirements covering the topology of jobs and data.

When designing the grid application architecture, some of the key items to consider are:

- Where grid jobs have to or can run
- How to distribute and deploy them over a network
- How to package them with essential data
- Where to store the executables within the network
- How to determine a suitable node for executing the individual jobs

The following are some factors that should be included in the consideration of the above items:

- Location of the data and its access conditions for the job
- Amount of data to be processed by the jobs
- Interfaces needed for any interaction with certain devices
- Inter-process communication needed for the job to complete its tasks
- Availability and performance values of the individual nodes at time of execution
- Size of the job's executable and its ability to be moved across the network

When developing grid-enabled applications you may not know anything about the topology of the grid on which they will run. However, especially in the case of an intra-grid that may be put in place to support a specific set of applications, this information may be available to you. In such a case, you may want to structure

your application and grid in such a way as to optimize the environment by considering the location of the resources, the data, and the set of nodes that a particular application might run on.

3.8 Passing of data input/output

As defined earlier, any job in the grid application needs to pass data in and out in the sense of a data producer and a data consumer.

There are various ways to realize the passing of data input and output that are to be considered during application architecture and design:

- *Command line interface* (CLI) can be a natural way for batch jobs and standard applications to receive data. In this case, the data input normally will not be complex in nature, but consists of certain arguments used as parameters to control the internal flow of the job. Such CLIs can easily be integrated in scripts executed at the system level or within a given interpreter. The transfer of data to the job as a consumer happens immediately at launch time. The amount of data will normally be small. For larger amounts of data there can be arguments that specify the name of a data file or other data source.
- *Data store* of any kind, such as data files in the file system (local or on a LAN or WAN) or records in a database, a data warehouse or other storage system that is available. These data stores can be used for input as well as output of data given that the required access rights are granted to the job. The transfer of data in can be done anytime before the job executes, and likewise the output data could be read anytime after the job completes, therefore providing flexibility for data movement operations.
- *Message queues*, like those provided by WebSphere MQSeries®, are well suited to be used for asynchronous tasks within a grid application, especially when guaranteed delivery of the data provided to the job and generated by the job is of high importance. A job can access the data queues in various ways, normally using specific APIs for putting or getting data as well as for polling the queue for data waiting for processing. In an environment where message queueing servers are already installed, this type of data passing may be desirable.
- *System return value*, is a corresponding case to the CLI and normally a way a batch job or any CLI invoked program will return data, or at least status information about how the job ended. This indicates to the grid server or grid portal the status of the individual job and requires appropriate management. The resulting data of the job may be passed to a data store or message queue for further processing or presentation.

- *Other APIs*, when communicating with Web services, Web servers, application servers, news tickers, measurement devices, or any other external systems, the appropriate conditions for data passing in and out have to be taken into consideration. In these cases, you may use HTTP, HTML, XML, SOAP, or other high-level protocols or APIs.

As indicated, for a grid application there may not be only one way to pass data for a job, but you may use any combinations of the described mechanisms. It is advised to program grid jobs in such a way that the data sources and sinks are generically handled for more flexible grid topologies. The optimal solution depends on the environment and the requirements to be considered at the architecture and design phase of the grid application.

3.9 Transactions

Handling of transactions in their strict definition of commit and roll-back is not yet well suited for a common grid application. The OGSI does not cover these services. However, a grid application may include subsystems or launch transaction-aware operations to subsystems such as CICS®.

The handling of transactions within a grid application easily becomes quite complex with the given definitions, and it needs to be carefully applied. The added benefits of a grid application may be outweighed by the complexity while implementing transactions.

The future development of the OGSA standard may include transaction handling as a service, though at the moment there is no support.

3.10 Data criteria

Any application, at its core is processing data. This means that we must take a closer look at data being used for and within a grid application. A detailed discussion is provided in Chapter 4, “Data management considerations” on page 71.

Data influences many aspects of application design and deployment and determines whether a planned grid application can provide the expected benefits over any other data solution.

3.11 Usability criteria

While much of a grid computing solution is involved with infrastructure and middle ware, it is still appropriate to consider aspects of the solution that relate to usability.

3.11.1 Traditional usability requirements

Traditional usability requirements address features that facilitate ease-of-use with the system. These features address interaction, display, and affective attributes that provide users with an effective, responsive, and satisfactory means to use the system. Hence, these features must be also be addressed when developing a grid computing solution; in other words, this is “business as usual” and continues to play an important part in establishing the requirements for a grid solution.

Usability requirements are used to:

- Provide baseline guidance to the user interface developers on user interface design.
- Establish performance standards for usability evaluations.
- Define test scenarios for usability test plans and usability testing.

Some of the typical usability requirements established for an IT solution play a role and include:

- *Tailorability*: What requirements exist for the user to customize the interface and its components to allow optimization based on work style, personal preferences, experience level, locale, and national language?
- *Efficiency*: How will the application minimize task steps, simplify operations, and allow end-user tasks to be completed quickly?

3.11.2 Usability requirements for grid solutions

Grid solutions must address usability requirements recognizing a variety of user categories that may include:

- End users wishing to: Log in to the grid, submit applications to the grid, query status, and view results
- Owners/users of donor machines
- Administrators and operators of the grid

Consequently, the typical steps followed to identify these requirements for any solution should continue to be followed when creating a grid solution. In addition, the following items may influence the design of grid solutions.

Installation

Ease of installation should provide automatic installation by a non-technical person rather than a systems programmer with the need to modify scripts, recompile software, and so on. The install process should be equally straightforward for host, management, and client nodes regardless of the potentially heterogeneous nature of the nodes in terms of operating system or configuration.

Unobtrusive criteria

Transparency and ease of use, as well job submission and control are not obvious items, but are essential for a good grid design.

- The use of a grid should be transparent to the user. The grid portal should isolate the user from the need to understand the makeup of the grid.
- Is documentation available or required for all categories of user including executive level summaries on the nature and use of the grid, programmer and administrative support staff? Where possible, the documentation should provide demos and examples for use.
- Ease of resource enrollment after any installation steps should provide simple configuration of grid parameters to enable the node and its resources to be a participant on the grid. The administrator of the grid or user of a donor machine should not require special privileges to enroll.
- Ease of job submission should alleviate the need for the user to understand the makeup of the grid, search for available resources, or have to provide complex parameters other than from the business nature of the application. It may be appropriate to provide multiple channels for job submission including command line (although this has not typically provided ease-of-use) and a graphical user interface via the grid portal.

  If the architectures of the grid resources are heterogeneous in nature, the solution should provide automation to hide these complexities and provide tools for compiling applications for multiple execution environments. This could also be considered under portability requirements typically addressed under the non-functional requirements.
- Ease of user and host access control should be provided from a single source with appropriate security mechanisms.

Informative and predictable aspects

Status of the grid must be readily available to continually show the status and operation of the grid. This may include indicators showing grid load or utilization, number of jobs running, number of jobs queued but not yet dispatched, status of hosts, available resources, reserved resources, and perhaps highlighting bottlenecks or trouble spots.

Since the makeup of the grid may be changing dynamically, predicting response times becomes harder. The appropriate trade-offs should be discussed to establish acceptable requirements with associated costs based on the needs of the business.

Resilience and reliability

Some aspects for resilience and reliability of the grid application have already been covered. In this section it is highlighted from the grid user perspective.

- Particular attention must be paid to the requirements for handling failures. Failures should be handled gracefully. The nature of the application must be understood to identify the correct handling of failures and to provide automatic recovery/restart where possible. Appropriate user notification should be included, recognizing that the actual user may not always be connected to the grid. Consequently, asynchronous mechanisms for feedback might need to be incorporated.
- The nature of applications that are suitable to run on the grid may provide a level of tolerance to failure not typically found in traditional applications. An example of this maybe in the “scavenging” scenario where the application as a whole may be able to tolerate failure of one or more sub-jobs. Since jobs are run on donor machines, the application is subject to the availability of these machines, which are typically outside the application’s scope of control. Consequently, the application must tolerate not receiving results from jobs dispatched to these donor machines.
- Applications must be fully integrated with systems management tools to report status and failures. In addition, requirements should be established for how this information will be made available to the end user indicating the status of their jobs.
- Consideration may also be given to providing intermediate results to an end user when these can provide valid results.

3.12 Non-functional criteria

There are several non-functional requirements that influence grid application architecture and have to be addressed up front.

An important topic is licensing in a grid environment. Licensing covers software licenses that are required for running the whole or parts of the grid application.

From the user perspective, performance plays a role. This is especially important when opening the grid for broad use. This often means unpredictable workload that needs to be taken care of during design of the application.

Finally, grid application development is a topic to be covered before code development and implementation can be started.

3.12.1 Software license considerations

One question that commonly arises when discussing grid computing is that of software license management. There are many products and solution designs that can help with license management.

Commercial software licenses

It is important to discuss how to deal with software licenses that are used inside the grid. Insufficient numbers of licenses may seriously hinder the expansion or even exclude certain programs or applications from being used in a grid environment.

The latter is the case if the grid wants to access personally licensed applications on a personal computer, for example, in a scavenging mode use of single-user licensed software. This cannot be done without violation of the license agreement.

Different models

The range of license models for commercial software spans from all restrictive to all permissive.

Between these two extremes there are numerous models in the middle ground, where licenses are linked to a named user (personal license), a workgroup, a single server, or a certain number of CPUs in a cluster, to a server farm, or linked to a certain maximum number of concurrent users and others.

Software licenses are given with a one-time charge or on a monthly license fee base. They can include updates or require purchase of new licenses. All this varies from vendor to vendor, and from customer situation to customer situation depending on individual agreements or other criteria.

Software licenses may allow for the migration of software from one server to another or may be strictly bound to a certain CPU. Listing all possible software licensing models could easily fill a book, but we cover a few below.

Service Provider License Agreement

Subscriber Access Licenses (SALs) are offered by service providers, for example, on a pay-per-use basis or as a flat rate for a certain maximum number of access times per month/week/year.

IT service providers in turn may acquire software licenses from ISVs for use by their customers, or they may simply host software, for which the end user will pay directly to the providing ISV according to their agreed license model.

Open source licensing

Another complexity is added when a software product is built that contains or requires open source software like the Globus Toolkit or Apache WebServer. The open source model is based on the principle that anybody (an ISV or private person) provides software to any interested parties, that can be modified, customized, or improved by the recipient.

The modifying recipient in turn can offer this changed code to anybody, who again can change it when needed. So there can be many developers in a loose community participating in development and improvement of a given set of code.

In this case, licenses are not bound to binary executables but cover source code as well. The following three licensing models for open source software are the most common, though there are several more, which may need to be investigated in any specific case.

BSD, MIT, Apache (all permissive licenses)

The license models for BSD, MIT, and Apache are all permissive, which means that they allow for free distribution, modification, and license changes. Software without copyright (public domain software) falls under this category as well.

For details on BSD licensing see:

`http://www.opensource.org/licenses/bsd-license.php`

For MIT licenses see:

`http://www.opensource.org/licenses/mit-license.php`

For the Apache Software License see:

`http://www.opensource.org/licenses/apachepl.php`

LGPL (persistent license)

The Lesser General Public License (LGPL) allows free distribution of the software, but restricts modifying it. All derivative work must be under the same LGPL or GPL. The definition of this license type can be found at:

http://www.opensource.org/licenses/lgpl-license.php

GNU GPL, IBM Public License (persistent and viral license)

The GNU General Public License (GPL) as well the IBM Public License (PL) shows a persistent and viral model, which means that it allows free distribution and modifying, but all bundled and derivative work must be under GNU GPL as well.

The GNU GPL can be found at either of the following Web sites:

http://www.gnu.org/copyleft/gpl.html
http://opensource.org/licenses/gpl-license.php

The IBM PL can be found at:

http://www.opensource.org/licenses/ibmpl.php

For Open Source Initiative (OSI) certified licenses and approvals visit:

http://opensource.org/docs/certification_mark.php

For the OSI portal simply go to:

http://www.opensource.org

There is a list of all approved open source licenses at the following Web site. GPL, LGPL, BSD and MIT are the most commonly used so-called “classic” licenses.

http://www.opensource.org/licenses/

License management tools

In order to manage most of these license models in a network there are a number of license management tools available. These tools assure that all software that is included in a network or a grid application is properly used according to its license agreements.

Most of the license manager providers offer an SDK with APIs for various programming languages. The span of license models covered by each product varies. In the following some of the most often used tools are listed.

FLEXlm

In the Linux world there is foremost FLEXlm, which offers 11 core models and 11 advanced licensing models. The core models include: Node-locked, named-user,

package, floating (concurrent) over network, time-lined, demo, enable/disable product, upgrade versions and a few more.

The advanced licensing models span from capacity, over site license, license sharing (user, groups, hosts), floating over list of hosts, high-water mark, linger license, overdraft, and pay-per-use, to network segments and more.

The complete list of supported licensing can be found at the following Web site:

http://www.globetrotter.com/flexlm/lmmodels.sthm

More information about the use and advantages of this de facto standard of electronic license management technology in the Linux world is available at:

http://www.globetrotter.com/flexlm/flexlm.shtm

Tivoli License Manager

IBM Tivoli License Manager is a software product that supports management of licenses in a network. Due to its nature, it is possible to reflect most of the license models being used in the industry. IBM Tivoli License Manager can reflect various stages of use during a piece of software's life time.

The IBM Redbook *Introducing IBM Tivoli License Manager*, SG24-6888, provides examples of how to reflect IBM, Microsoft, Oracle, and other vendors' license models in its management.

IBM Tivoli License Manager is integrated with WebSphere Application Server and available for AIX, Solaris, and several Microsoft Windows platforms.

More details about the product are also given on the IBM Software Group Web site at:

http://www.ibm.com/software/tivoli/products/license-mgr/

IBM License Use Management (LUM)

IBM License Use Management (LUM) in its current version 4.6.6 is designed for technical software license management as it is deployed by most IBM use-based software products. It is intended to be integrated with any vendor software in order to control use-based licensing of the software.

LUM is available for all Windows platforms, AIX, HP-UX, Linux, IRIX, and Solaris. It supports a wide range of C, C++, and Java development environments. It can be used in networks with most of the available Web servers.

Software developers are enabled to reflect various use-based license models while integrating LUM APIs in their software products. It can be used for monitoring and controlling the use of software in networks.

More details are found on the IBM software group Web site at:

http://www.ibm.com/software/is/lum/

Platform Global License Broker

Among the various ISVs that offer grid software products, Platform shows a special grid-oriented license management feature named Platform Global License Broker.

This product runs on AIX, HP-UX, Compaq Alpha, and IRIX. It uses Globetrotter FLEXlm 7.1 as described in “FLEXlm” on page 64. More details on Platform Global License Broker is available on the Internet at:

http://www.platform.com/products/wm/glb/index.asp

General license management considerations

When designing and deploying grid-enabled applications it is important to understand any licensing requirements for required runtime modules. If designing a broker or utilizing MDS to identify possible target resources on which to run the application, the existence or applicability of any required software licenses should be taken into account.

3.12.2 Grid application development

In order to develop a grid application, the Globus Toolkit offers a broad range of services that are becoming more comprehensive with the next version. Included are commodity Grid Kits (CoG) for a number of programming languages and models, such as Java, C/C++, Perl, Python, Web services, CORBA, and Matlab (see http://www.globus.org/cog/ for details and updates).

Grid Computing Environment (GCE)

The Globus Toolkit, CoGs, and appropriate application development tools form a Grid Computing Environment capable of supporting collaborative development of grid applications. In context with the Globus initiative, various frameworks for collaborative and special industry solutions, as well as a grid services flow language are being worked on. For details and recent activities on Application Development Environments (ADEs) for grids at Globus refer to:

http://www.globus.org/research/development-environments.html

Examples of using the Java CoG for grid application development are given in Chapter 6, “Programming examples for Globus using Java” on page 133.

Grid-enabled Message Passing Interface (MPI)

A grid-enabled Message Passing Interface that fits with the Globus Toolkit is provided by MPICH-G2. This implementation of the MPI standard allows the

coupling of multiple machines and provides automatic conversions of messages. It addresses solutions that are *distributed by nature* as well those *distributed by design*. For details and the latest updates see:

http://www.niu.edu/mpi/

Grid Application Development Software (GrADS)

An example for a distributed-by-design scenario is given by the Grid Application Development Software project. The goal of this approach sponsored by the US Department of Energy (DoE) is to simplify distributed heterogeneous computing in the same way that the World Wide Web (WWW) simplified information sharing over the Internet.

GrADS has been developed for various Unix versions (Solaris, HP-UX, Linux). It is written in C/C++ and exploits LDAP. Several software projects are built on top of it, including a Common Component Architecture (CCA) and XCAT.

The GrADS project explores the scientific and technical problems that occur when applying grid technology for real applications in everyday life. Details on GrADS are found at:

http://nhse2.cs.rise.edu/grads/

IBM Grid Toolbox

For grid application development with Globus, the CoGs can be used with appropriate IDEs. IBM research offers the IBM Grid Toolbox as a set of development tools for grid application development on AIX and Linux. It supports most of the grid services (GRAM, GSI, MDS, GASS, simple CA, I/O, and so on) as described in this publication. Details and download of the IBM Grid Toolbox are available at:

http://www.alphaworks.ibm.com/tech/gridtoolbox

Grid Application Framework for Java

Another application development item recently offered by IBM research is the Grid Application Framework for Java (GAF4J). It abstracts the interface to the Globus Toolkit for Java programmers by introducing an abstraction layer on top of Globus. Details and downloads are available at:

http://www.alphaworks.ibm.com/tech/GAF4J

Other tools

When searching the Internet for “grid application development” one finds a large number of hits with most of them pointing to AD tool vendors who claim their tools as being ready for supporting grid application development. Even so, any comprehensive competitive analysis will not be up to date as it is published, because the standards (OGSA) are still developing. Grid computing evolves in

various directions for different purposes, and the application development tools market is constantly changing.

3.13 Qualification scheme for grid applications

In this section a usable format of a qualification scheme for grid applications is provided. We also provide a criteria list that may be looked at as a knock-out list. That is, it includes attributes of an application or its requirements that may inhibit an application from being a good candidate for a grid environment.

The list may not be complete and depends on the local circumstances of resources and infrastructure. The qualification scheme acts as a basis for architecture and project planning for a grid application.

3.13.1 Knock-out criteria for grid applications

Earlier sections have discussed considerations for grid enabling an application from the perspectives of infrastructure and application functionality. However, not all applications lend themselves to successful or cost-effective deployment on a grid. A number of criteria may make it very difficult, require extensive work effort, or even prohibit grid-enabling an application. Criteria below may preclude deploying an application to the grid without having to perform an extensive analysis of the application.

Some facts such as temporary data spaces, data type conformity across all nodes within the network, appropriate number of SW licences available in the network for the grid application, higher bandwidth, or the degree of complexity of the job flow can be solved, but have to be addressed up front in order to create a reasonable grid application.

An application with a serial job flow can be submitted to a grid, but the benefits of grid computing may not be realized, and the application may be adversely affected due to grid management overhead. However, by exploiting the grid and submitting the application to more powerful remote nodes it may very well provide business value.

In this list of knock-out criteria the most critical items are named that most certainly hinder or exclude an application from use on a grid:

1. High inter-process communication between jobs without high speed switch connection (for example, MPI, in general, multi-threaded applications need to be checked for their need of inter-process communication.
2. Strict job scheduling requirements depending on data provisioning by uncontrolled data producers.

3. Unresolved obstacles to establish sufficient bandwidth on the network.
4. Strongly limiting system environment dependencies for the jobs (see 3.5, “Job dependencies on system environment” on page 54).
5. Requirements for safe business transactions (commit and roll-back) via a grid. At the moment there are standards for secure transaction processing on grids.
6. High inter-dependencies between the jobs, which expose complex job flow management to the grid server and cause high rates of inter-process communication.
7. Unsupported network protocols used by jobs may be prohibited to perform their tasks due to firewall rules.

3.13.2 The grid application qualification scheme

The application architecture considerations and requirements of grid services lead to a qualification scheme, which highlights the solution requirements and criteria that impact building a grid application.

The scheme shown in Appendix A, “Grid qualification scheme” on page 297, provides a summary of 35 criteria, of which most will apply to any grid application, but not all. The criteria are to be seen in relation to each other and to the individual situation of the project.

The scheme is intended for use at the analysis phase of a grid application development project and allows the user to quickly detect and highlight the most critical issues for the grid application to be built. It may also reveal any show stoppers or identify more effort to be planned to solve a certain problem.

The scheme is provided as a tool that can be modified for specific use at a given grid application project.

3.14 Summary

The approach to build a grid-enabled application either from scratch or based on existing solutions adds a wide range of aspects for problem analysis, application architecture, and design. This chapter has provided an overview of the issues to consider for any grid application.

Some of these items may not apply for every project. Some aspects are familiar from other application development projects and are not elaborated on in depth. Others that are new aspects due to the nature of a grid application are provided with greater detail.

The grid qualification scheme in Appendix A, “Grid qualification scheme” on page 297, represents a summary of most of the essential items to consider. It is meant to be a base document to be used during the analysis phase of a grid project.

In the next chapter, we discuss considerations specific to data management.

4

Data management considerations

No matter what the application, it generally requires input data and will produce output data. In a grid environment, the application may submit many jobs across the grid and each of these jobs in turn will need access to input data and will produce output.

One of the first things to consider when thinking about data management in a grid environment is management of the input data and gathering of the output data. If the input data is large and the nodes that will execute the individual jobs are geographically removed from one another, then this may involve splitting the input data into small sets that can be easily moved across the network assuming the individual jobs need access to only a subset of the data.

The splitting of input data and the joining of output data from the jobs is often handled by a wrapper around the job that handles the splitting dynamically when the job is submitted and retrieves the individual data sets after each job has completed.

The second aspect of data management is during the job execution by itself. The job needs to access data that may not be available on local storage. Several solutions are available:

- Data is stored on network-accessible devices and jobs work on the data through the network.

© Copyright IBM Corp. 2003. All rights reserved.

- Data is transferred to the execution node before the job is executed such that the job can access the data locally.

4.1 Data criteria

Any application, at its core, is processing data. This means that we must take a closer look at data being used for and within a grid application. The following sections cover criteria related to handling data when deciding whether an application is a good candidate for a grid.

Data influences many aspects of application design and deployment and determines whether a planned grid application can provide the expected benefits over any other data solution. As the grid can be dynamically set up and changed, there are some special data-related considerations.

The following sections describe several considerations related to data in the grid, such as the distribution and location of data in regard to accessing jobs and when and how data is created and consumed by jobs.

4.1.1 Individual/separated data per job

Any job will work on a specified set of input data. The data sources and sinks can be of various kinds. The following are some questions to be considered:

- Can the data be separated for individual use by a defined job?

  It is important that each single job receives a well-defined set of input data and has an equally well-defined place to leave its computing results.

- Is the data replicated in such a way that it is available at any time the assigned job needs to run or rerun?

  This means that we must be careful about changes to the data sets a grid job has to work with. One way of solving it can be to establish certain local and temporary data caches that live as long as the grid application runs. These data caches are under the control of the grid server or grid portal. These caches can act as data sources for more than one job, for example, if multiple jobs use the same data but perform different actions. It may be especially important if one job is launched redundantly, or the output of one job determines the input for another job.

- Can a separatable data space be created for any job of the grid application?

  A question of how to assure each job's data does not interfere with any other job or processes being executed anywhere on the grid.

- Are there interdependencies between jobs in a grid application that require synchronization of the data?

  This may require certain locks on data for read or write access. It also means that we must consider how failures while producing data are to be solved among any dependent jobs.

4.1.2 Shared data access

Related to the separation of data for individual jobs is the question of sharing data access with concurrent jobs and other processes within the network. Access to data input and the data output of the jobs can be of various kinds. The following considerations are kept generic so that they can be applied to the actual cases appropriately.

During the planning and design of the grid application, you must consider whether there are any restrictions on the access of databases, files, or other data stores for either read or write. The installed policies need to be observed and, depending on the task the job has to fulfill, sufficient access rights have to be granted to the jobs.

Another topic is the availability of data in shared resources. It must be assured that at run-time of the individual jobs the required data sources are available in the appropriate form and at the expected service level.

Potential data access conflicts need to be identified up front and planned for. You must ensure that individual jobs will not try to update the same record at the same time, nor dead lock each other. Care has to be taken for situations of concurrent access and resolution policies imposed.

Federated databases

If a job must handle large amounts of data in various different data stores, you may want to consider the use of federated databases. They offer a single interface to the application and are capable of accessing data in large heterogeneous environments.

Federated databases have been developed in regards to data-intensive tasks in the life sciences industry for drug discovery, genome search, and so on. In these cases, the federated databases are the central core of a data grid installation.

Federated database systems own the knowledge about location (node, database, table, record, for instance) and access methods (SQL, VSAM, or others, perhaps privately defined methods) of connected data sources. Therefore, a simplified interface to the user (a grid job or other client) requires that the essential information for a request should not include the data source, but rather use a discovery service to determine the relevant data source and access method.

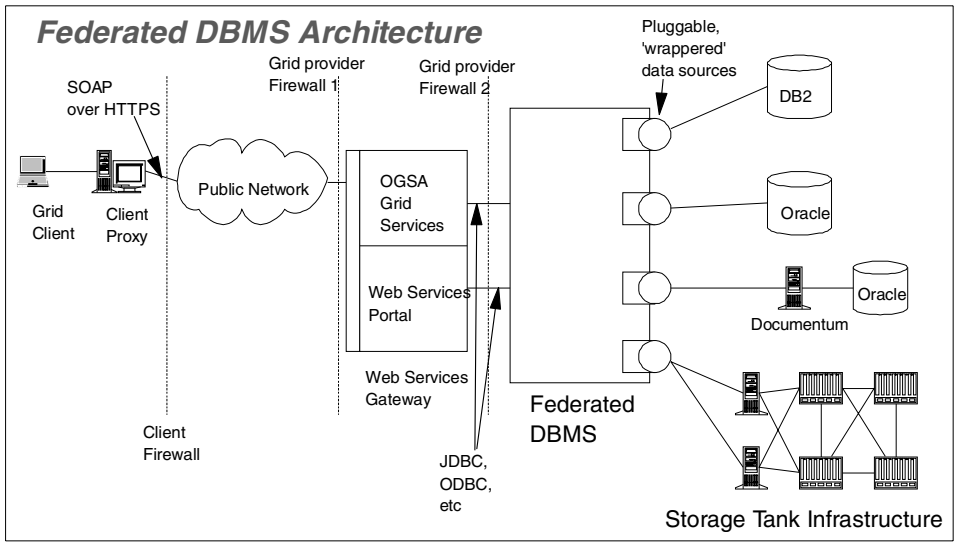


Figure 4-1 Federated DBMS architecture

The use of such a federated database solution can also be considered as part of a more general grid application, where the jobs access data by acting as clients of a federated database.

Additionally, as shown in Figure 4-1, the use of Storage Tank™ technology for large data store capacities can be included and managed by federated databases.

IBM data management products for grid applications

There are several IBM products that support the federated database concept, such as DB2® Federated Server, DB2 Data Joiner, DB2 Discovery Link, DB2 Relation Connect, DB2 Information Integration, and many more.

Additionally, there are several white papers, products, solution offerings, and related material available from the IBM DB2 Web sites. Please see the following Web site for more details and support:

http://www.ibm.com/software/data/

4.1.3 Locking

In a grid context locking is also important. Read/write locking is well understood as in any other concurrency situation with databases and other data sources. Read only locks for keeping accuracy of data sets should be considered, too.

4.1.4 Temporary data spaces

Within grid applications temporary data spaces are often needed. During planning of the grid application the forms and amount of temporary data space should be considered.

Points to consider include:

- *Availability of sufficient data space* for the amount of data a job or the federated system requires. Also, caches managed by the grid server or grid portal should be considered.
- *OS-specific requirements* for data spaces, data access, and management need to be taken care of, especially if the job-specific data needs to be or can be local to the job, or whether cross-system, network, or platform data access has to be planned. The format, access, and locking of data can vary, if not indirectly accessed.
- *Local or shared file system-dependent requirements* are to be considered to assure for optimal runtime access.
- *Memory for temporary data* of a job can vary from system to system, as a node may run several jobs in parallel and share the memory for many processes. In order to allow the best performance and avoid unnecessary data swapping, the memory requirements of the jobs are important to understand. In the case of compiled executables, there may be different memory needs depending on the compiler and operating system it is compiled for.

4.1.5 Size of data

Knowing, separating, and compiling the amount of data within a grid application is important. The total amount of data includes all data used for input and output of all jobs within the grid application.

Note that this total amount of data may exceed the amount of data input and output of the grid application, as there can be a series of sub-jobs that produce data for consumption of other sub-jobs and so forth, until finally the resulting data of the application are produced.

For permanent storage the grid user needs to be able to locate where in the grid the required storage space is available. Other temporary data sets that may need to be copied from or to the client also need to be considered.

4.1.6 Network bandwidth

The amount of data that has to be transported over the network can be restricted by available bandwidth. Less bandwidth requires a rather careful planning of the expected data traffic within a grid application at runtime.

Compression and decompression techniques are useful to reduce the data amount to be transported over the network. But in turn, it raises the issue of consistent techniques on all involved nodes. This may exclude the utilization of scavenging for a grid, if there are no agreed standards universally available.

The central question is: What bandwidth is needed to allow all required input and output data of the jobs to be transported over the network?

4.1.7 Time-sensitive data

Another issue to be covered in this context is time-sensitive data. Some data may have a certain lifetime, meaning its values are only valid during a defined time period. The jobs in a grid application have to reflect this in order to operate with valid data when executing.

Especially when using data caching or other replication techniques, the currency of the data used by the jobs needs to be assured at any given point in time.

As discussed in 3.2.2, “Serial flow” on page 47, the order of data processing by the individual jobs, especially the production of input data for subsequent jobs, has to be observed.

4.1.8 Data topology

The issues discussed above about the size of the data, network bandwidth, and time sensitivity of data determine the location of data or the topology of the data.

Depending on the job, the following data-related questions need to be considered:

- Is it reasonable that each job or set of jobs accesses the data via a network?
- Does it make sense to transport a job or set of jobs to the data location?
- Is there any data access server (for example, implemented as a federated database) that allows access by a job locally or remotely via the network?
- Are there time constraints for data transport over the network, for example, to avoid busy hours and transport the data to the jobs in a batch job during off-peak hours?

- Is there a caching system available on the network to be exploited for serving the same data to several consuming jobs?
- Is the data only available in a unique location for access, or are there replicas that are closer to the executable within the grid?

These questions refer to input as well as output data of the jobs within the grid application.

Data topology graph

In order to answer these questions a graphical representation can help, like the one in Figure 4-2. This data topology graph lists all available nodes on one axis and all the jobs of the application on the other axis. All required data stores are then placed on the appropriate intersections.

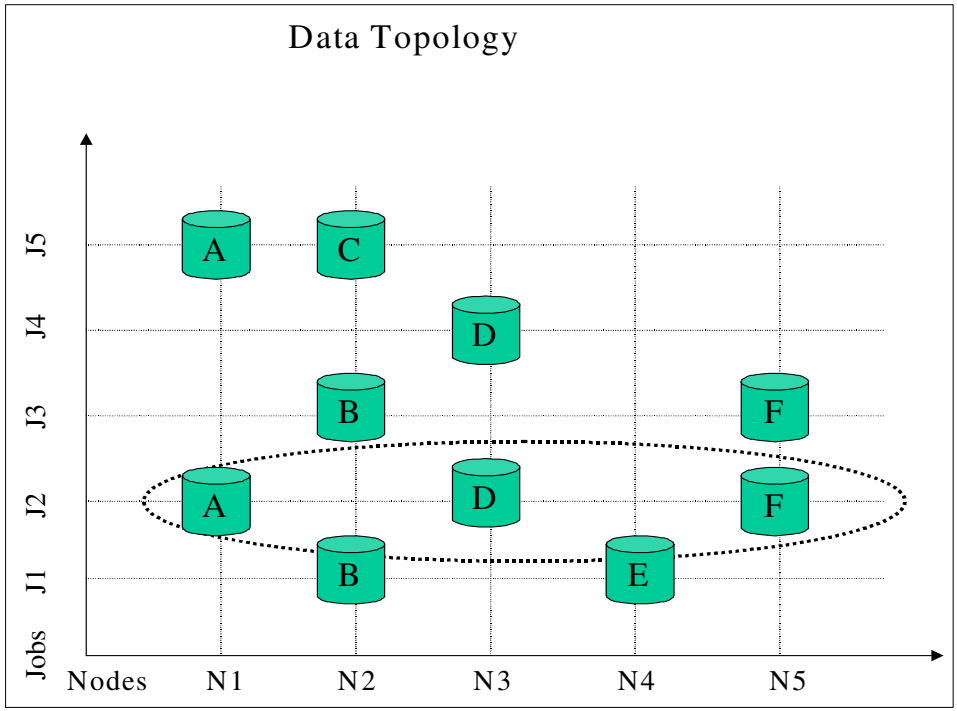


Figure 4-2 Data topology of a grid

The example in Figure 4-2 reveals that job J2 has to access data of three different data sources, which are located on different nodes in the network. In this case it is necessary to check whether the data extract of each of the data sources A, D, and F that is needed for job J2 can be sent over the network to the node where job J2 is going to be executed.

Depending on the nature of the data sources, the essential data for job J2 may be extracted or replicated to be close to or on the job executing node. In case the data cannot be separated and the data amount is large, it is necessary to check whether the job can be split into individual jobs or sub-jobs to be executed close to the data.

If this is not possible one might consider moving the data of A, D, and/or F to a single node where job J2 can run.

The data topology graph helps to identify needs for data splitting and replication.

4.1.9 Data types

When considering writing jobs for a grid application that could run on any system anywhere in the world, the question of data types, code pages, and trans-coding arises. For example, when transferring a C-source file containing the following statement written by a German programmer as:

```
{*argv[1]='\0'}
```

It may appear as:

```
æ*argvÆ1Å='Ø0'å
```

On a Danish system or as:

```
&|argv(1)/'=0'*
```

On an American system, where the compiler would not understand it. Therefore, one should be aware of and take into account the type of data, its representation, format, and standards for data exchange.

To name a few of the standards and variations that might be used or have to be considered within the application:

- ASCII vs EBCDIC
- Single-byte vs double-byte character sets
- Unicode (UTF-8, -16, -32)
- Big endian vs little endian
- APIs and standards for data exchange
  - SOAP
  - MQ
  - SQL
  - HTML
  - XML
  - J2EE
  - JDBC
  - And more

- Different multi-media formats for
  - Images
  - Animation
  - Sound
  - Fonts
  - Archives
  - And more
- Measurement units
  - Metric vs non-metric
  - Currencies
  - And more

4.1.10 Data volume and grid scalability

The ability for a grid job to access the data it needs will affect the performance of the application. When the data involved is either a large amount of data or a subset of a very large data set, then moving the data set to the execution node is not always feasible. Some of the considerations as to what is feasible include the volume of the data to be handled, the bandwidth of the network, and logical interdependences on the data between multiple jobs.

Data volume issues

In order to use a grid application, transparent access to its input and output data is required. In most cases the relevant data is permanently located on remote locations and the jobs are likely to process local copies. This access to the data means a network cost and it must be carefully quantified.

Data volume and network bandwidth play an important role in determining the scalability of a grid application.

Data splitting and separation

As indicated in 4.1.8, “Data topology” on page 77, the data topology considerations may require the splitting, extraction, or replication of data from involved data sources in order to allow the grid to properly function and perform.

There are two general cases that are suitable for higher scalability in a grid application: Independent tasks per job and a static input file for all jobs.

Independent tasks

A suitable case for a grid-enabled application is when the application can be split into several jobs that are able to work independently on a disjunct subset of the input data. Each job produces its own output data and the gathering of all of the results of the jobs provides the output result by itself. Figure 4-3 on page 81 illustrates this case.

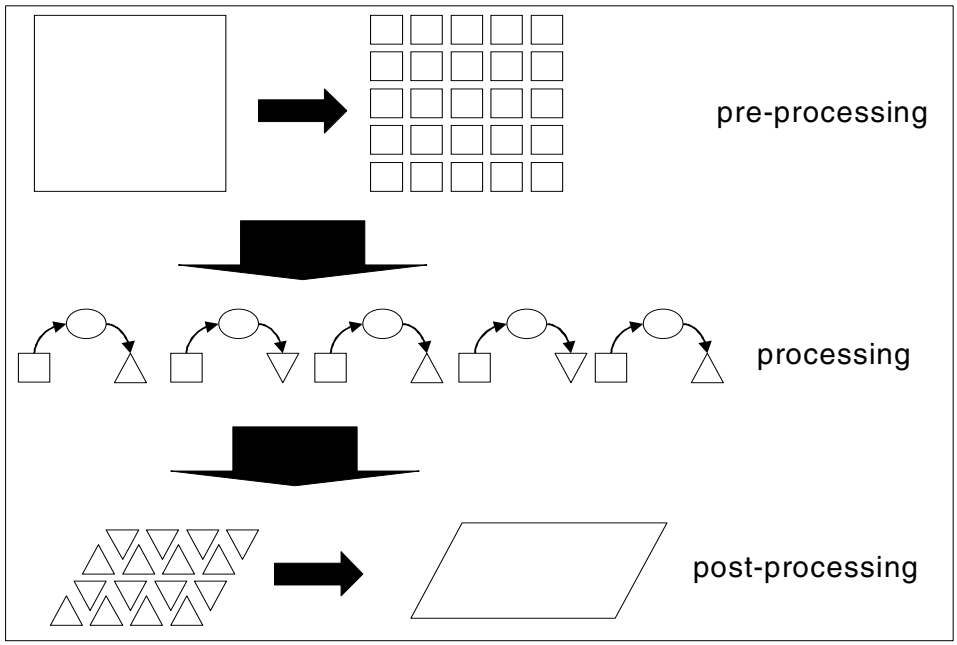


Figure 4-3 Independently working jobs on disjunct data subsets

This specific case can be easily integrated in a Globus grid environment.

The scalability of such a solution depends on the following criteria:

- Time required to transfer input data
- Processing time to prepare input data and generate the final data result

In this case the input data may be transported to the individual nodes on which its corresponding job is to be run. Preloading of the data might be possible depending on other criteria like timeliness of data or amount of the separated data subsets in relation to the network bandwidth.

Static input files

Static input files is the other case that may be suited to using an application on a grid. Figure 4-4 on page 82 illustrates how in this case each job repeatedly works on the same static input data, but with different “parameters,” over a long period of time.

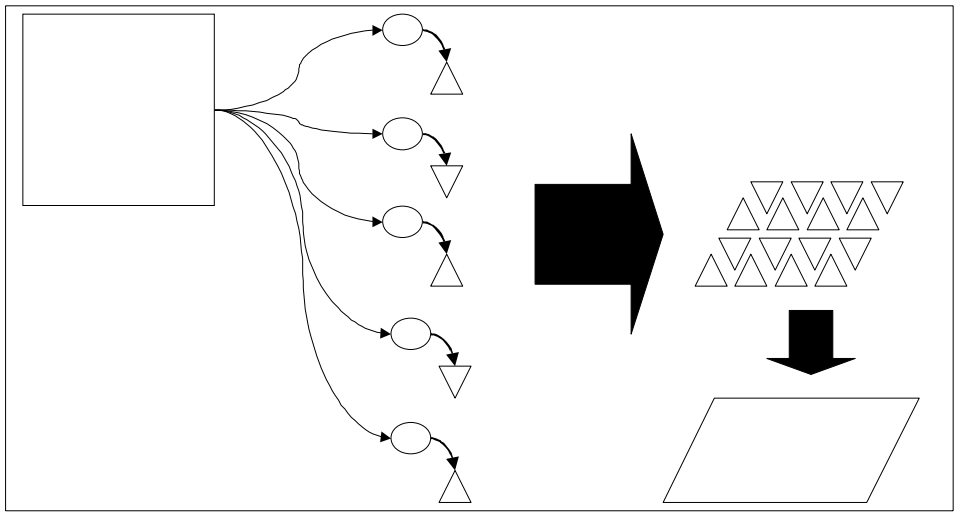


Figure 4-4 Static input data processed by jobs with changing parameters

In this case, the job can work on the same static input data several times but with different parameters, for which it generates differing results.

A major improvement for the performance of the grid application may be derived by transferring the input data ahead of time as close as possible to the compute nodes.

Other cases of data separation

More unfavorable cases may appear when jobs have dependencies on each other. The application flow may be carefully checked in order to determine the level of parallelism to be reached.

The number of jobs that can be run simultaneously without dependences is important in this context. In this section, a few cases are discussed in more detail from the data perspective.

For independent jobs, there needs to be synchronization mechanisms in place to handle the concurrent access to the data. The Globus Toolkit does not provide any synchronization mechanisms to manage these dependencies and, therefore, these cases need to be managed by the grid application developers. However, Globus-core modules provide portable mutex, condition variables, and thread implementations that help to implement such mechanisms.

Synchronizing access to one output file

This case is shown in Figure 4-5 on page 83. Here all jobs work with common input data and generate their output to be stored in a common data store.

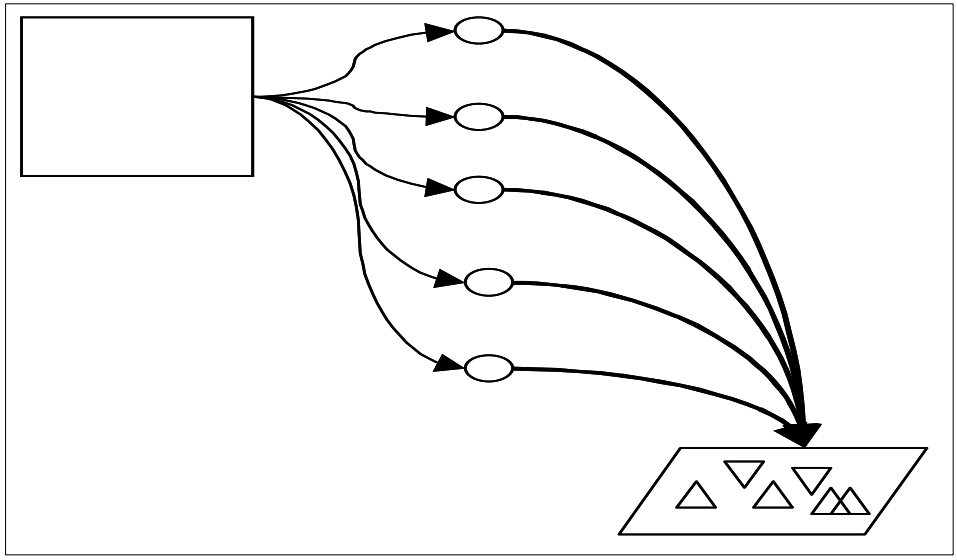


Figure 4-5 All jobs works on the same data and write on the same data set

The output data generation implies that software is needed to provide synchronization between the jobs. Another way to process this case is to let each job generate individual output files, and then to run a post-processing program to merge all these output files into the final result.

A similar case is illustrated in Figure 4-6 on page 84. Here each job has its individual input data set, which it can consume. All jobs then produce output data to be stored in a common data set. Like described above, the synchronization of the output for the final result can be done through software designed for the task.

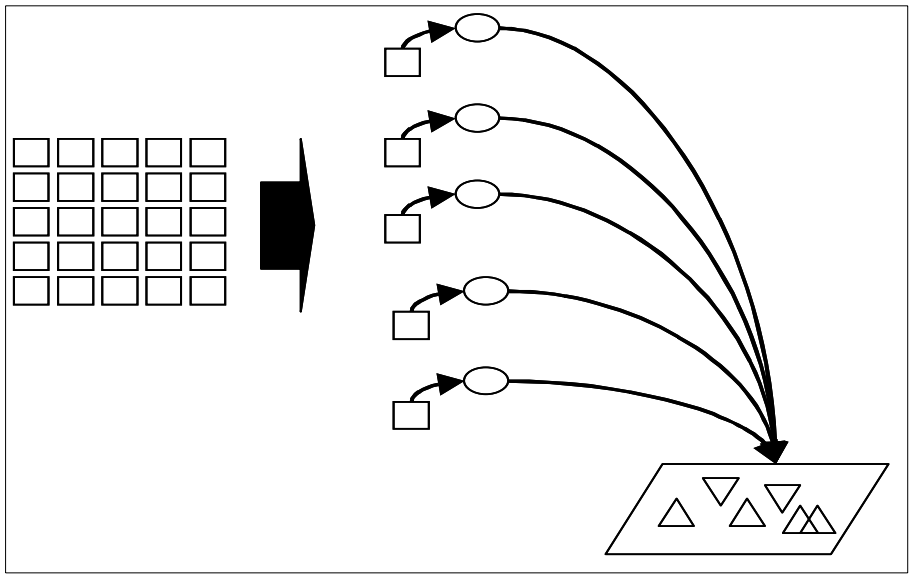


Figure 4-6 Jobs with individual input data writing output into one data store

Hence, thorough evaluation of the input and output data for jobs in the grid application is needed to properly qualify it. Also, one should weigh the available data tools, such as federated databases, a data joiner, and related products, in case the grid application to be built becomes more data oriented or the data to be used shows a complex structure.

4.1.11 Encrypted data

Data encryption is mentioned in this context in order to complete this section of this publication. A rather in-depth discussion of the topic is given in *Introduction to Grid Computing with Globus*, SG24-6895.

At the architecture and design stage of a grid application project it is important to cover the encryption issues that are required by the solution and the customer. The subjects to consider are authentication, access control, data integrity, data confidentiality, and key management. For a grid application this can be addressed via a Public Key Infrastructure (PKI) or via the Grid Security Infrastructure (GSI) as supported by Globus.

For a grid application, the Certificate Authority (CA) for public keys as well the various encryption mechanisms (symmetric or asymmetric) can be used. During the architecture and design phases, one needs to determine which CA and which encryption mechanism to use.

It has to be assured that the appropriate infrastructure is implemented and reflected in the grid application to be built. Hence, this is a topic for the qualification scheme (see 3.13.2, “The grid application qualification scheme” on page 69) used at the early stages of a grid project.

4.2 Data management techniques and solutions

A grid can increase application performance by way of parallelism. This implies that a big job must be divided into smaller ones. From a data point of view, it may be necessary to split the input data and to gather the results after processing. The two operations that occur respectively before and after the job submission are called data pre-processing and data post-processing. The data splitting can be triggered each time a job is submitted or it can done one time in advance. Similarly, the data gathering and joining of results can be handled multiple ways, depending in the requirements.

In the first case, the Globus Toolkit does not provide tools to perform the pre- and post-processing tasks. Therefore, software will need to be developed to perform the two tasks. Shell script and scripting languages like Perl or Python may be appropriate to perform these tasks, depending on the type of data store and the size of the data. It may be mandatory to use languages like C/C++, which produce compiled executables to achieve acceptable performance.

In the second case, the data will remain distributed on different locations for all jobs that will process this data. Therefore, users need to have a logical view of this file distributed across a set of nodes. This logical view will be provided by a catalog, whereas each storage node will store the different parts of the file. The Globus Toolkit provides a framework to manage this case: It provides an LDAP schema to implement the replica catalog, as well as a C/C++ API to access and manage this information.

The user of a grid environment needs transparent access to its input and output data. This data will most of the time be permanently located on remote locations and the job will process local copies only. The transparent access to the data has a network cost and it must be carefully qualified. Data access transparency also requires that the storage resources be sufficient, and this also needs to be qualified.

4.2.1 Shared file system

Sharing data across the compute nodes may sometimes be mandatory or may appear as the simplest solution to permit a computation to be distributed. When data are in plain files, network file systems are a convenient solution. The question is not to choose between staging the data in and out or using a shared

file system, but to find the appropriate data flow that will provide optimal performance. Therefore, a mixed solution can be considered. For example, a network file system could be shared across a cluster of compute nodes, and input and output files would be staged in and out of the shared files system from a permanent storage center.

Globus Toolkit does not provide a shared file system but can be used with any available shared file system. Therefore, in 4.2.11, “Global file system approach” on page 90, we describe in detail some shared file system solutions available today or in the near future.

4.2.2 Databases

Data grids have generally focused on applications where data is stored in files. However, databases have a central role in data storage, access, organization, authorization, and so on for numerous applications.

Globus Toolkit 2.x provides no direct interface for relational or object databases such as DB2, Oracle, and MySQL. However, a grid-enabled application could certainly use any available API, such as SQL, to access these databases. However, there are a few things to consider:

- The GSI authentication mechanisms cannot be used if a program needs to connect to a database.
- The Globus Toolkit 2.x does not provide an API to manipulate databases.
- By default, there is no information on databases that can be retrieved from the MDS. Nevertheless, you can create your own information provider. See:

  http://www-unix.mcs.anl.gov/~slang/mds_iprovider_example/

The Database Access and Integration Services Working group (DAIS-WG http://www.gridforum.org/6_DATA/dais.htm) is currently working on an implementation for such a database service for the Globus Toolkit V3. Several projects are currently working on related issues.

4.2.3 Replication (distribution of files across a set of nodes)

Data replication is an optimization technique well known in the distributed systems and database communities as a means of achieving better access times to data. Data replication is an optimization technique. The key concepts are:

- A registration operation that adds information about files on a physical storage system to an existing location and logical collection entries. Hence, new files can be made available to users by registering them in existing locations and collection entries (lists of files).

- The replication operation copies a file to storage systems that are registered as locations of the same logical collection and updates the destinations' location entries to include the new files.
- The publishing operation takes a file from a storage system that is not represented in the replica catalog, copies the file to a destination storage system that is represented in the replica catalog, and updates the corresponding location and logical collection entries.

4.2.4 Mirroring

For safety and performance reasons, data are usually mirrored across a set of nodes. This way several access points are provided for jobs that need to process this data and data, brokering can be used to determine which access point needs to be used according to various criteria such as the subnet where the job is run, or user identification. Mirroring consists of being able to synchronize data manipulations that occur at different locations. The mirroring can be synchronous or asynchronous (mirroring happens at certain time intervals).

The Globus Toolkit does not provide mirroring capabilities, but the European DataGrid project and the Particle Physics Data Grid project developed the Grid Data Mirroring Package on top of the Globus Toolkit.

4.2.5 Caching

Caching provides temporary storage on the execution nodes and avoids network access during job execution. The primary purpose of the cache is efficiency. Programs and any prerequisite modules that are required to invoke a job, as well as input data, are good candidates to be stored locally on each execution node. A suitable case is when a job needs to process the same data multiple times (maybe each run with different parameters). However, using a cache is not the only solution, and considerations such as transfer times and space requirements should be taken into account.

The Globus Toolkit implements cache mechanisms for files only and not for data files. Globus GASS cache provides a C API that provides for the manipulation of the cache. The Globus Toolkit also provides the **`globus-gass-cache`** command to manipulate the contents of a local or remote GASS cache.

4.2.6 Transfer agent

The role of the transfer agent is to provide speed and reliability for files being transferred. These files can be:

- Executables, scripts, or other modules representing the programs that will be run remotely
- Job dependencies, for example, dynamic shared libraries
- Input files
- Output or results files

The Globus Toolkit uses the GridFTP protocol for all file transfers. This protocol is detailed in “GridFTP” on page 194. File transfer is built on top of a client/server architecture that implies that a GridFTP server must be running on the remote node to be able to transfer a file to the remote host. The globus-io module and Globus GASS subsystem transparently use the GridFTP protocol. Note that the GSIssh transfer tool, gsiscp, does not use the GridFTP protocol, but uses the same encrypted flow transfer used by openSSH (http://www.openssh.org).

4.2.7 Access Control System

There is no component in the Globus Toolkit that provides an enforcement of Access Control List policies. Each administrator, by configuring the grid-mapfile stored on its resources and file’s user access rights, can allow or disallow the remote job execution on its resources under a certain user ID. It can only enforce local policy.

A project still under development, the Community Authorization Service (CAS), should provide such access control. The administrator of a resource server grants permissions on a resource to the CAS server. The CAS server grants fine-grained permissions on subsets of the resources to members of the community. For more information, see the Community Authorization Service (CAS) site:

http://www.globus.org/security/CAS/

4.2.8 Peer-to-peer data transfer

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization. Each node of the peer-to-peer can be both client and server. For example, when a client begins to download a file from a server, it allows other clients to start downloading the same file from its own storage.

There is no peer-to-peer solution currently provided by the Globus Toolkit. However, a group at the Globus Forum is working on this domain (relation of OGSA/Globus and Peer2Peer). A complete report should be provided for GGF8. or more information, see the following Web site:

http://www.gridforum.org/4_GP/ogsap2p.htm

4.2.9 Sandboxing

For performance reasons, runtime files tend to be stored on the local storage where the job will use them. Programs and data files are stored on a remote site and copied to local disks when needed.

The performance of the LAN environment may be good enough so that a network file system could provide the needed bandwidth, and therefore could avoid the overload of data transfer. This becomes not true in a WAN environment, or if the jobs need to work repeatedly on the same data sets.

A sandbox provides a temporary environment to process data on a remote machine and *limited* access to the resources of the node. This way, the job execution cannot interfere with normal processes running on this node. Data are copied into this sandbox. The sandbox can be encrypted so that any other applications normally running on the node could not access the job data.

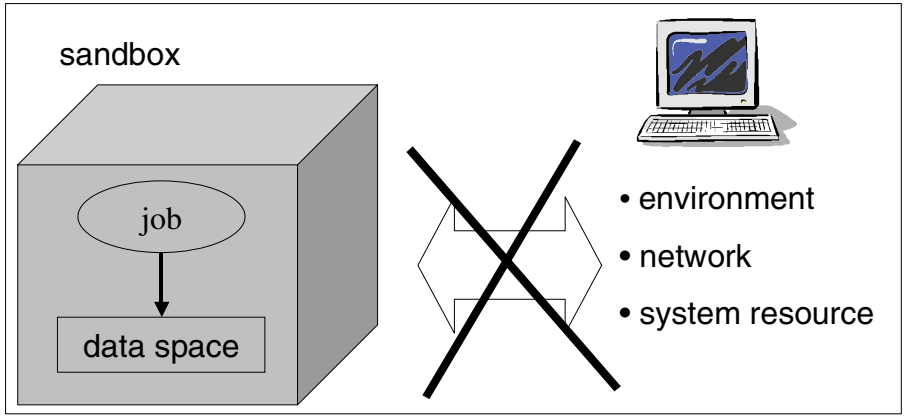


Figure 4-7 Sandboxing

Globus Toolkit also provides **globus-gass-cache** commands to manipulate the contents of a local or remote GASS cache. Each entry in a GASS cache consists of a URL local file name, a list of tags, and a reference count for each tag. When the last tag for a URL is removed, the local file is removed from the cache. The cache directory is actually a directory located in the .globus/.gass-cache directory of the user under which the job is executed. The GASS cache is

transparently used during job invocation via GRAM: Files specified in the RSL strings are put into the cache if they are referenced as URLs. See “globus-gass-cache” on page 193 for a more complete description.

4.2.10 Data brokering

A storage broker may be used by applications to provide them with the appropriate storage resources. It must provide the following capabilities:

- Searching for an appropriate data storage location. This means querying the Replica Catalog for all physical locations and querying each physical location.
- Matching the resources according to the application needs.
- Accessing the data.

The Globus Toolkit 2 does not provide a storage broker engine. However, some implementations have been written that use GRIS, GridFTP, and the Replica Catalog available in the Globus Toolkit 2.2.

4.2.11 Global file system approach

A global file system can be easily integrated into a Grid solution based on the Globus Toolkit. A global file system provides access to storage, and any applications can use POSIX system calls to access files without the need of any grid-specific APIs.

Several solutions exist today that will fit project expectations across various criteria: Performance, cost, ease of deployment, and so on. However, they should not be considered as the only alternative. Global file systems are usually suitable for cluster needs (where a cluster is defined as a set of nodes interconnected by a high-performance switch) or in LAN environment. Nevertheless, Global file systems are often unique to one organization and therefore cannot be easily shared by multiple organizations.

Network File System (NFS)

NFS is almost universally used in the Unix world and is the de facto standard for data file sharing in a LAN environment. NFS V2 supports files up to a maximum size of 2 GB. NFS V3 improves file transfer performance and gets rid of some of the NFS V2 limitations (64-bit file support, write caching). NFS uses the udp protocol, but can also use tcp protocol as it does by default under AIX.

NFS V4 is the emerging standard for UNIX file system access. It will be supported in the AIX operating system and in the forthcoming Linux 2.6 kernel. NFS V4 includes many of the features of AFS® and DFS™. NFS V4 uses strong Kerberos V5 security and Low Infrastructure Public Key, and it should also

perform on a WAN environment as well as it does on a LAN by using file caching and minimizing the number of connections needed for read and write operations.

NFS V4 appears to be a good alternative to AFS and DFS file systems, and could be used in a grid environment where a cost-effective, shared file system is required.

For more information on NFS Version 4 Open Source Reference Implementation see:

http://www.citi.umich.edu/projects/nfsv4

For NFS V4 for the ASCI project see:

http://www.citi.umich.edu/projects/asci

General Parallel File System

GPFS allows shared access to files that may span multiple disk drives on multiple nodes. GPFS is currently supported for Linux and AIX operating systems. A high-performance inter-connect switch such as Myrinet or a SP switch is mandatory to achieve acceptable performance.

GPFS is installed on each node as a kernel extension and appears to jobs as just another file system. This implies that the jobs only need to call normal I/O system calls to access the files.

The GPFS advantages are:

- The jobs still use standard file system calls.
- The jobs can concurrently access files from different nodes with either read or write I/O calls.
- Increases bandwidth of the file system by striping across multiple disks.
- Balances the load across all disks to maximize throughout.
- Supports large amounts of data.

As all nodes in a grid cannot be connected to the same high-performance network, GPFS is not the ultimate solution for the grid but is a good solution when local file sharing is requested on a local cluster that will process grid jobs. GPFS is also a good candidate for the permanent storage of very large files that need to be partially copied to other nodes on the grid by using the Globus Toolkit.

Avaki Data Grid solution

Avaki Data Grid provides a solution for sharing files across Wide Area Networks. Its two main features are:

- It provides an NFS interface for applications that can therefore transparently access files stored in the Avaki files system. For security reasons, the Avaki Data Grid is usually mounted locally.

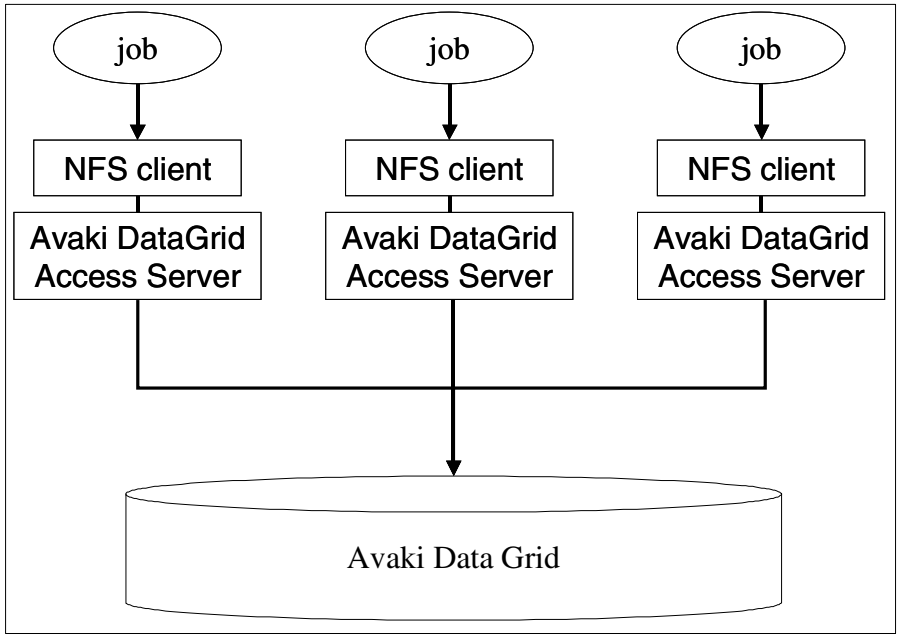


Figure 4-8 Accessing Avaki Data Grid through NFS locally mounted file system

- By creating an Avaki share, you can map local files on a node into the Avaki Data Grid. This way, the files become available to all nodes connected to the Avaki Data Grid. The synchronization between the local files and their Avaki DataGrid copies occurs periodically, based on a configuration option. For example, every three minutes.

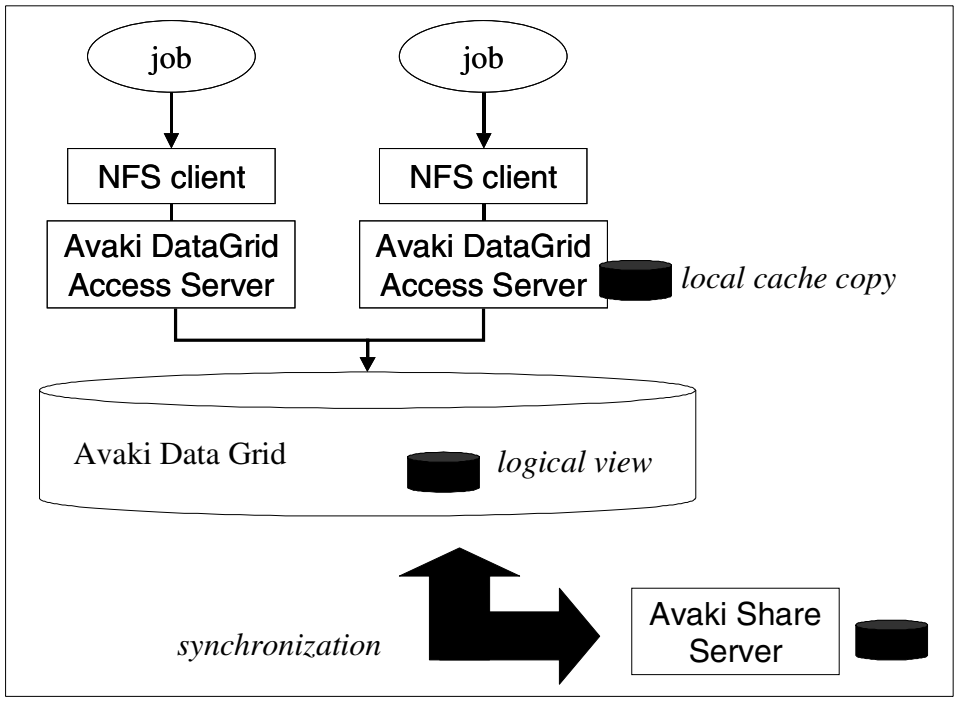


Figure 4-9 Avaki share mechanism

Avaki also provides complete user management and Access Control List policies. For applications, Avaki maps Avaki user authorization with local operating system user authorization. Avaki can also be tied into an existing network user authentication system like LDAP so that information does not need to be duplicated into a separate grid access control list.

For more information see:

http://www.avaki.com

4.2.12 SAN approach

Storage Area Networks (SAN) are well suited for high-bandwidth storage access. When transferring large blocks, there is not much processing overhead on servers since data is broken into a few large segments. Hence, SAN is effective for large bursts of block data. It can be used when very large files (for example, videos) have to be manipulated and shared at a level of reliability that no ordinary network can support.

Storage Tank

IBM provides a complete storage management solution in a heterogeneous, distributed environment. Storage Tank is designed to provide I/O performance that is comparable to that of file systems built on bus-attached, high-performance storage. In addition, it provides high availability, increased scalability, and centralized, automated storage and data management.

Storage Tank uses Storage Area Network (SAN) technology that allows an enterprise to connect thousands of devices, such as client and server machines and mass storage subsystems, to a high-performance network. On a SAN, heterogeneous clients can access large volumes of data directly from storage devices using high-speed, low-latency connections. The Storage Tank implementation is currently built on a Fibre Channel network. However, it could also be built on any other high-speed network, such as Gigabit Ethernet (iSCSI), for which network-attached storage devices have become available.

Storage Tank clients can access data directly from storage devices using the high-bandwidth provided by a Fibre Channel or other high-speed network. Direct data access eliminates server bottlenecks and provides the performance necessary for data-intensive applications.

An installable file system (IFS) is installed on each IBM Storage Tank client. An IFS directs requests for metadata and locks to an IBM Storage Tank server and sends requests for data to storage devices on the SAN. Storage Tank clients can access data directly from any storage device attached to the SAN.

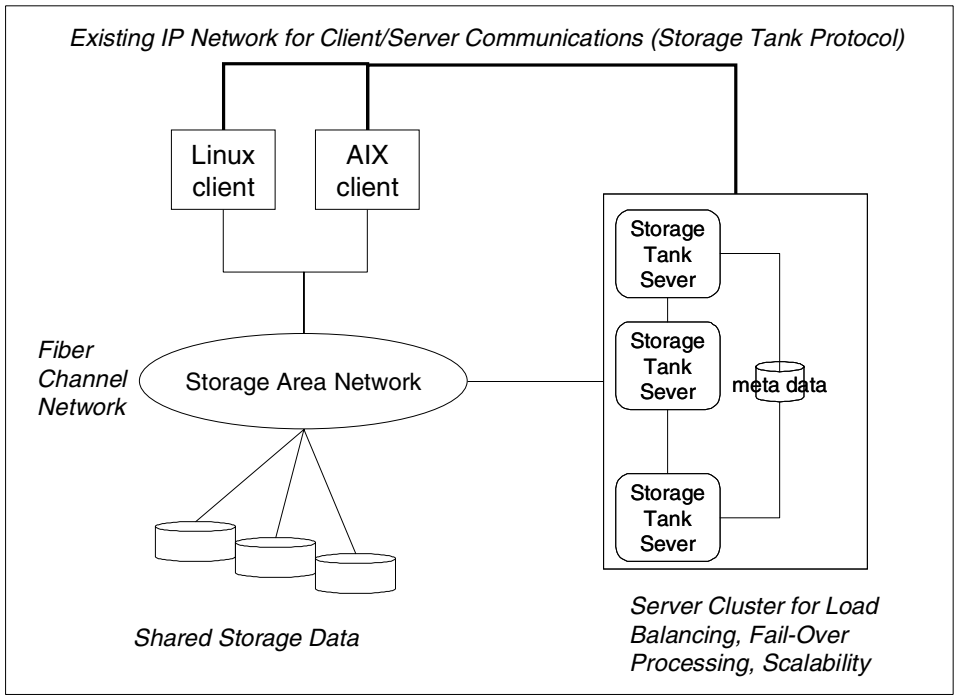


Figure 4-10 Storage Tank architecture

The Global File System (GFS)

GFS allows multiple servers on a Storage Area Network to have read and write access to a single file system on shared SAN devices. GFS is IBM certified on its xSeries™ servers only and for the Linux operating system.

GFS can support up to 256 nodes.

For more information see:

http://www.sistina.com/products_gfs.htm

4.2.13 Distributed approach

Another approach to managing data needs in a grid is to distribute the data across the grid nodes through processes such as replication or mirroring. The following sections describe these approaches in more detail.

Replica Catalog

The Globus Toolkit Replica Catalog can keep track of multiple physical copies of a single logical file by maintaining a mapping from logical file names to physical locations. A *replica* is defined as a “managed copy of a file.”

The catalog contains three types of objects:

- The collections that are a group of logical names.
- The locations that contain information required to map between logical name and the multiple locations of the associated replicas. Each location represents a complete or partial copy of a logical collection on a storage system. The location entry explicitly lists all files from the logical collection that are stored on the specified physical storage system.
- The logical file entry that is an optional object to store attribute-value pairs for each individual file. They are used to characterize each individual file. Logical files have globally unique names that may have one or more physical instances. The catalog may contain one logical entry in the Replica Catalog for each logical file in a collection.

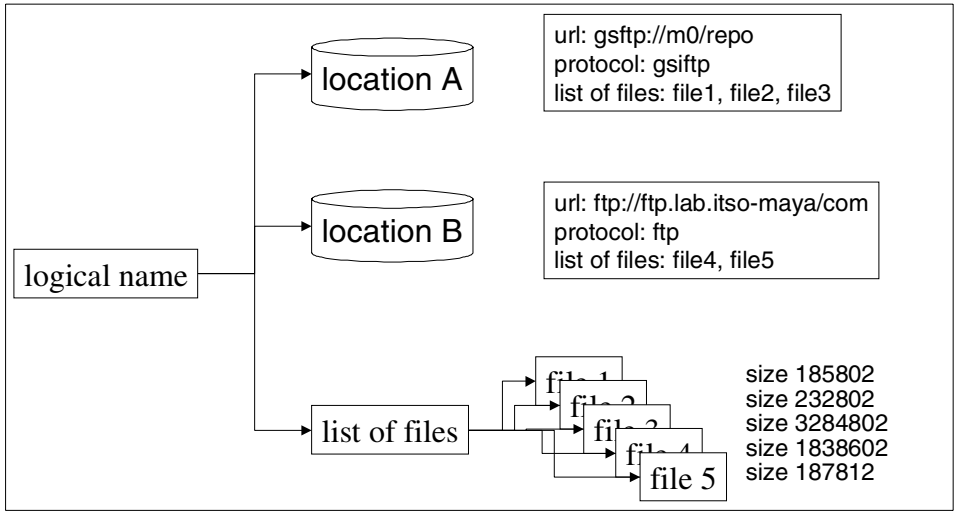


Figure 4-11 Replica logical view

Replica Catalog functions can be used directly by applications by using the C/C++ APIs provided by Globus. They provide the following operations:

- Creation and deletion of collection, location, and logical file entries
- Insertion and removal of logical file names into collections and locations
- Listing of the contents of collections and locations
- A function to return all physical locations of a logical file

Examples are provided in “Replication” on page 208 by using shells commands provided by the Globus Toolkit 2.2.

Replica Location Service (RLS)

The Replica Location Service is a new component that appears in Globus Toolkit 2.4. This component maintains and provides access to information about the physical locations of replicated data. This implementation was co-developed by the Globus Project and Work Package 2 of the European DataGrid project. RLS is intended to eventually replace the Globus Toolkit's Replica Catalog component. For more information see:

http://www.globus.org/rls
http://www.isi.edu/~annc/RLS.html

Grid Data Mirroring Package (GDMP)

The GDMP is client-server software developed in C++ and built on top of the Globus Toolkit 2 framework. Every request to a GDMP server is authenticated by the Globus Security Infrastructure. It provides two things:

- a generic file replication tool to replicate files from one site to one or more remote sites. A storage location is considered to be a disk space on a single machine or several machines connected via a local area network and a network file system.
- GDMP manages Replica Catalog entries for file replicas and therefore makes the file visible to the grid. Registration of user data into the Replica Catalog is also possible via the Globus Replica Catalog C/C++ API.

The concept is that data producer sites publish their set of newly created files to a set of one or more consumer sites. The consumers will then be notified of new files entered in the catalog of the subscribed server and can make copies of required files, automatically updating the Replica Catalog if necessary.

Note 1: Files managed by GDMP should be considered as read-only by the consumer.

Note 2: GDMP is not restricted to disk-to-disk file operation. It can deal with files permanently stored in a Mass Storage System.

GDMP C++ APIs for clients provide four main services:

- Subscribing to a remote site for obtaining information when new files are created and made public
- Publishing new files and thus making them available and accessible to the grid

- Obtaining a remote site's file catalog for failure and recovery
- Transferring files from a remote location to the local site

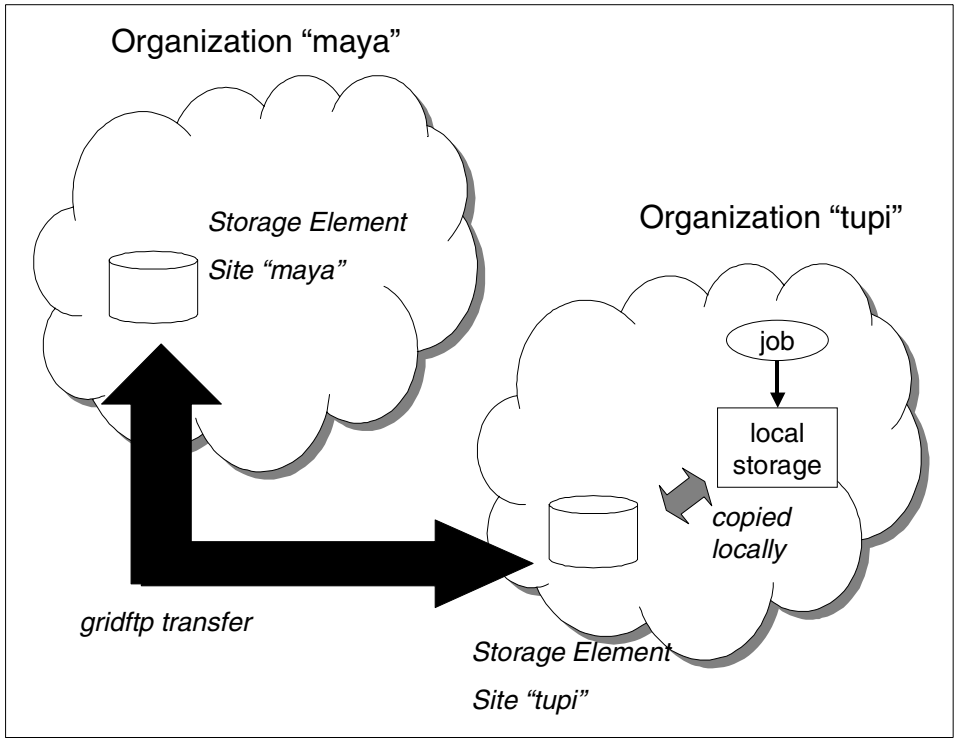


Figure 4-12 File replication in a data grid between two organizations

GDMP is available at:

http://project-gdmp.web.cern.ch/project-gdmp

4.2.14 Database solutions for grids

As covered in 4.2.2, "Databases" on page 86, the Globus Toolkit V3 should provide a set of services to access data stored in databases and several projects that are ongoing, such as SpitFire in EU DataGrid and the UK Database Task Force, can already be tested.

Until these solutions become ready, several commercial solutions can help to enable database access in a grid application.

Federated databases

A federated database technology provides a unified access to diverse and distributed relational databases. It provides transparency to heterogeneous data sources by adding a layer between the databases and the application.

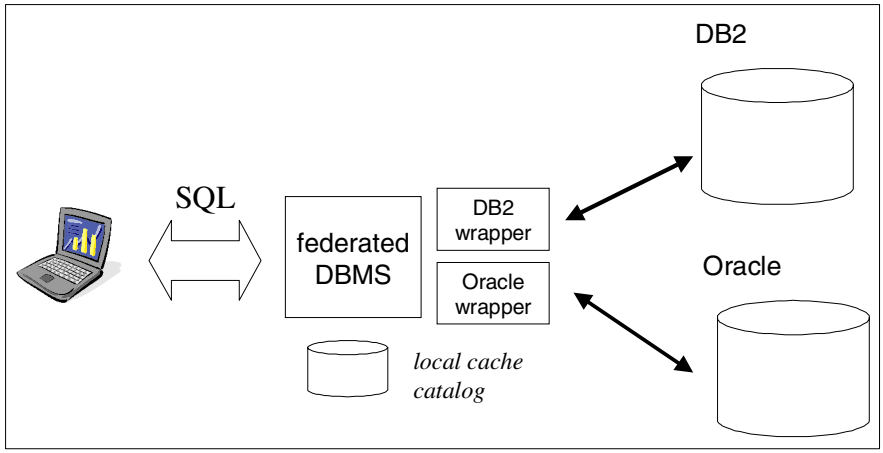


Figure 4-13 Federated databases

In a federated database, each data source is registered to the federated DBMS along with its wrapper. A wrapper is a piece of code (dynamic library) loaded at runtime by the federated database to access a specific data source. The application developer only needs to use a common SQL API (like ODBC or JDBC) in their applications and to access the federated database.

The developer also needs to explicitly specify the data source in the federated query. Consequently, the application must be changed when new data sources are added.

Currently, no federated databases use the Globus Toolkit 2.2 Security Infrastructure (GSI) to authenticate or authorize the query. The application developer needs to manage the authentication process to the database apart from the Globus Security API.

IBM DB2 Connect™ provides a solution to transparently access remote legacy systems using common database access APIs like ODBC and JDBC.

OGSA Database Access and Integration

The Open Grid Services Architecture Database Access and Integration (OGSA-DAI) is a project conceived by the UK Database Task Force and is working closely with the Global Grid Forum DAIS-WG and the Globus team.

The project is in place to implement a general grid interface for accessing grid data sources like relational database management systems and XML repositories, through query languages like SQL, XPat, and XQuery. XQuery is a new query language like SQL and under draft design in the W3C.

The software deliverables of the OGSA-DAI project will be made available to the UK e-Science community and will also provide the basis of standards recommendations on grid data services that are put forward to the Global Grid Forum through the DAIS working group. For more information, see:

http://umbriel.dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI/

Spitfire

Spitfire is a project of the European DataGrid Project. It provides a grid-enabled middleware service for access to relational databases, providing a uniform service interface, data and security model, as well as network protocol. Spitfire uses the Globus GSI authentication and thus can be easily integrated in an existing Globus infrastructure. Spitfire is currently using MySQL and PosrtGreSQL databases, and the Web services alpha release should be available soon.

Currently it consists of the Spitfire Server module and the Spitfire Client libraries and command line executables. Client-side APIs are provided for Java and C++ for the SOAP-enabled interfaces. The C++ client is auto generated from its WSDL description using gSOAP, which is an open-source implementation protocol. The gSOAP project is also used for the C implementation of the Globus Toolkit V3. For more information see:

http://www.cs.fsu.edu/~engelen/soap.html

Three SOAP services are defined: A Base service for standard operations, an Admin service for administrative access, and an Info service for information on the database and its tables.

Spitfire is still a beta project. For more information, see:

http://spitfire.web.cern.ch/Spitfire/

4.2.15 Data brokering

One data brokering solution available today is the Storage Resource Broker.

Storage Resource Broker

The Storage Resource Broker (SRB) developed by the San Diego SuperComputer Center is not part of the Globus Toolkit but can use its GSI PKI authentication infrastructure. Consequently, an SRB and a Globus grid can

coexist with the same set of users. SRB brings to Globus the ability of submitting metadata queries that permits a transparent access to heterogeneous data sources. The SRB API does not use the globus-io API, nor Globus gass, or GridFTP.

The SRB is a middleware that provides a uniform interface for connecting to heterogeneous data resources over a network and accessing replicated data sets. SRB permits an application to transparently access logical storage resources whatever their type may be. It easily manages data collection stored on different storage systems but accessed by applications via a global name space. It implements data brokering used by grid applications to retrieve their data.

The SRB consists of three components:

- The metadata catalog (MCAT)
- SRB servers
- SRB clients

The MCAT is implemented using a relational database such as Oracle, DB2, PostGresSQL, or Sybase. It maintains a Unix name space (file name, directories, and subdirectories) and a mapping of each logical name to a set of physical attributes and a physical handle for data access. The physical attributes include the host name and the type of resource (Unix File system, HPSS archive, database, and so on). The MCAT server handles requests from the SRB servers that may be information queries as well as instructions for metadata creation and update.

SRB, in conjunction with the Metadata Catalog (MCAT), provides a way to access data sets and resources based on their attributes rather than their names or physical locations.

Each data stored in SRB has a logical name that can be used as a handle for data operation. The physical location of data is logically mapped to the data sets that may reside on different storage systems. A server manages/brokers a set of storage resources. The supported storage resources are: Mass Storage system such as HPSS, UniTree, and DMF; and ADSM as file systems.

SRB provides an API for grid application developers in the following programming languages: C/C++, Perl, Python, and Java. For management purposes, SRB also provides a set of Unix shell commands as well as a GUI application and a Web application.

SRB supports the GSI security infrastructure that permits the integration of the SRB into the Globus Toolkit environment. (See http://www.npaci.edu/DICE/security/index.html). The Authentication and

Integrity of Data library (libAID) needs to be installed to permit SRB to use GSI authentication. LibAID provides an API to GSI. For more information, see the following Web site:

http://www.npaci.edu/DICE/SRB/

4.3 Some data grid projects in the Globus community

Many data-centric grid projects in the research community are based on the Globus Toolkit 2. They have developed various middleware to help handle the data management considerations. Here is a short list of large data grid projects whose middleware source codes are available.

4.3.1 EU DataGrid

The DataGrid project is a project funded by the European Union that aims to enable access to geographically distributed data servers. It is based on the Globus Toolkit 2.2 and therefore uses the Globus Data Grid framework: GridFTP protocol and replica management.

This project implements a middleware layer between applications and the Globus Toolkit 2.

The Grid Data Mirroring Package (GDMP) is a file replication tool that replicates files from one site to another site. It can manage replica catalog entries for file replicas. Note that all files are assumed to be read only. GDMP is a collaboration between the EU DataGrid and the Particle Physics Data Grid Project (PPDG). GDMP is described in detailed in “Grid Data Mirroring Package (GDMP)” on page 97. For more information see:

http://www.eu-datagrid.org

4.3.2 GriPhyn

The GriPhyN Project is developing grid technologies for scientific and engineering projects that must collect and analyze distributed, petabyte-scale data sets. GriPhyN research will enable the development of Petascale Virtual Data Grids (PVDGs) through its Virtual Data Toolkit (VDT). Virtual data means that data does not necessarily have to be available in a persistent form but is created on demand and then materialized when it is requested.

The Virtual Data Toolkit (VDT) is a set of software that supports the needs of the research groups and experiments involved in the Griphyn project. It contains two types of software:

- Core grid software: Condor Scheduler, GDMP (REF), and the Globus Toolkit. In future releases, VDT will use the NMI software (gsissh, kerberos/GSI gateway, Condor-G).
- Software developed to work with virtual data: Chimera is the first software of this kind.

The Chimera Virtual Data System (VDS) provides a catalog that can be used by application environments to describe a set of application programs ("transformations"), and then track all the data files produced by executing those applications ("derivations"). Chimera contains the mechanism to locate the "recipe" to produce a given logical file, in the form of an abstract program execution graph. These abstract graphs are then turned into an executable DAG for the Condor DAGman meta-scheduler by the Pegasus planner, which is bundled into the VDS code release. For more information, check the following Web site:

http://www.griphyn.org/

4.3.3 Particle Physics Data Grid

The Particle Physics Data Grid collaboration was formed in 1999. The purpose of this long-term project is to provide a data grid solution supporting the data-intensive requirements of particle and nuclear physics.

PPDG is actively participating in the International Virtual Data Grid Laboratory (iVDGL http://www.ivdgl.org) together with GriPhyN as a three-prong approach to data grids for US physics experiments. PPDG focuses on file replication and job scheduling. Also, it is working closely with complementary data grid initiatives in Europe and beyond: Global Grid Forum, European DataGrid, and as part of the HENP. For example, the Grid Data Mirroring Package has been a mutual effort of EU DataGrid and PPDG. For more information, check the following Web site:

http://www.ppdg.net/

4.4 Summary

A grid application must carefully take into account the topology of the data that will be processed during the job execution. Data can be centralized or distributed across the execution nodes. A mixed solution is usually the most appropriate, and it highly depends on the existing infrastructure.

There are several existing and evolving technologies that can be used to manage and access data in a grid environment, and we have described a few projects that have built tools on top of Globus to provide the required capabilities for data-oriented grids.

5

Getting started with development in C/C++

In this chapter, we will start looking at how these components are actually used, both through the command line and through programs written to the Globus APIs.

Since the Globus Toolkit ships with C bindings, we start out by providing some information for C/C++ programmers that will help them better understand the programming environment. We then provide some C/C++ examples of calling Globus APIs. The examples we use are based on the GRAM module for submitting jobs, and the MDS modules for finding resources.

© Copyright IBM Corp. 2003. All rights reserved.

5.1 Overview of programming environment

In the next three subsections we provide information about programming and building C/C++ applications that utilize the Globus Toolkit. For more details, and a list of all of the available APIs, please visit the Globus Web site.

5.1.1 Globus libc APIs

The Globus Toolkit 2.2 is a cross-platform development framework and allows the application development of portable grid applications by using its API.

The globus-libc API provides a set of wrappers to several POSIX system calls. The grid developer must use these wrappers to ensure thread-safety and portability. The globus equivalents to the POSIX calls add the prefix globus_libc to the function while prototypes remain identical. For example, globus_libc_gethostname() should be used instead of gethostname(), or globus_libc_malloc() instead of malloc().

Reference information is available at:

http://www.globus.org/common/globus_libc/functions.html

The globus-thread API provides system call wrappers for thread management. These include:

- thread life-cycle management
- mutex life-cycle, locking management
- condition variables, signal management

This API must be used to manage all asynchronous or non-blocking Globus calls and their associated callback functions. Usually a mutex and a condition variable are associated for each non-blocking Globus function and its callback function.

For this publication, we created a sample C++ object ITSO_CB whose source code is available in “ITSO_CB” on page 315. The method of this class actually uses the globus-thread APIs and can be used as an example of how to do so. This class is used in most of our examples.

Reference information for the thread-specific APIs is available at:

http://www.globus.org/common/threads/functions.html

5.1.2 Makefile

globus-makefile-header is the tool provided by the Globus Toolkit 2.2 to generate platform- and installation-specific information. It has the same functionality as the well-known autoconf tools.

The input parameters are:

- The flavor you want for your binary: gcc32, gcc32dbg for debugging purposes, gcc32pthr for multi-thread binary. The flavor encapsulates compile-time options for the modules you are building.
- The list of modules that are used in your application and that need to be linked with your application are globus_io, globus_gss_assist, globus_ftp_client, globus_ftp_control, globus_gram_job, globus_common, globus_gram_client, and globus_gass_server_ez.
- the --static flag can be used to get a proper list of dependencies when using static linking. Otherwise, the dependencies are printed in their shared library form.

The output will be a list of pairs (VARIABLE = value) that can be used in a Makefile as compiler and linker parameters. For example:

```
GLOBUS_CFLAGS = -D_FILE_OFFSET_BITS=64 -O -Wall
GLOBUS_INCLUDES = -I/usr/local/globus/include/gcc32
GLOBUS_LDFLAGS = -L/usr/local/globus/lib -L/usr/local/globus/lib
GLOBUS_PKG_LIBS = -lglobus_gram_client_gcc32 -lglobus_gass_server_ez_gcc32
-lglobus_ftp_client_gcc32 -lglobus_gram_protocol_gcc32
-lglobus_gass_transfer_gcc32 -lglobus_ftp_control_gcc32 -lglobus_io_gcc32
-lglobus_gss_assist_gcc32 -lglobus_gssapi_gsi_gcc32
-lglobus_gsi_proxy_core_gcc32 -lglobus_gsi_credential_gcc32
-lglobus_gsi_callback_gcc32 -lglobus_oldgaa_gcc32 -lglobus_gsi_sysconfig_gcc32
-lglobus_gsi_cert_utils_gcc32 -lglobus_openssl_error_gcc32
-lglobus_openssl_gcc32 -lglobus_proxy_ssl_gcc32 -lssl_gcc32 -lcrypto_gcc32
-lglobus_common_gcc32
GLOBUS_CPPFLAGS = -I/usr/local/globus/include -I/usr/local/globus/include/gcc32
```

These variables are built based on the local installation of the Globus Toolkit 2.2 and provide an easy way to know where the Globus header files or the Globus libraries are located.

Consequently, the procedure to compile a Globus application is the following:

1. Generate an output file (globus_header in the example) that will set up all the variables used later in the compile phase.

   ```
   globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist
   globus_ftp_client globus_ftp_control globus_gram_job globus_common
   globus_gram_client globus_gass_server_ez globus_openldap > globus_header
   ```

2. Add the following line in your Makefile to include this file:

   ```
   include globus_header
   ```

3. Compile by using make.

Example 5-1 Globus Makefile example

```
include globus_header

all:  SmallBlueSlave SmallBlueMaster SmallBlue

%.o: %.C
   g++ -c $(GLOBUS_CPPFLAGS) $< -o $@

SmallBlue:SmallBlue.o GAME.o
   g++ -o $@ -g $^

SmallBlueSlave:SmallBlueSlave.o GAME.o
   gcc -o $@ -g $^

SmallBlueMaster: GAME.o SmallBlueMaster.o itso_gram_job.o itso_cb.o
itso_globus_ftp_client.o itso_gassserver.o broker.o
   g++ -g -o $@ $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS)
```

Note: Sometimes the package may be not be available in all flavors. globus-makefile-header will only tell you that the package you required does not match the query, but will not inform you that it exists in another flavor.

The application will be linked with Globus static libraries or Globus dynamic libraries, depending on the kind of Globus installation you performed. You can use the shell command **ldd** under Linux to check if your application is dynamically linked to the Globus libraries located in $GLOBUS_LOCATION/lib.

Under Linux, if your application uses dynamically linked Globus libraries, then be sure that:

- Either the LD_LIBPATH_PATH is properly set to $GLOBUS_LOCATION/lib when you run your application
- Or $GLOBUS_LOCATION/lib is present in /etc/ld.so.conf

The list of main packages that are used in this publication are:

- globus_common used for all cross-platform C library wrappers
- globus_openldap for querying the MDS server
- globus_gass_server_ez to implement a simple GASS server
- globus_gass_transfer for GASS transfer
- globus_io for low-level I/O operation
- globus_gss_* for GSI security management
- globus_ftp_client, globus_ftp_control for gsiftp transfer
- globus_gram_job for job submission

5.1.3 Globus module

In Globus Toolkit V2, each Globus function belongs to an API provided by a specific Globus module. This module must be activated before any of the functions can be used. The globus_module API provides functions to activate and deactivate the modules:

- globus_module_activate() calls the activation function for the specified module if that module is currently inactive.
- globus_module_deactivate() calls the deactivation functions for the specified module if this is the last client using that module.

The function's return value is GLOBUS_SUCCESS if it was successful.

Example 5-2 Globus API module management

```
if (globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE) != GLOBUS_SUCCESS)
{
   cerr << " Cannot start GRAM module";
   exit(2);
};
int rc=globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
globus_assert(rc == GLOBUS_SUCESS);
globus_module_deactivate_all();
```

In the broker in “Broker example” on page 127, the GLOBUS_GRAM_CLIENT_MODULE is activated and deactivated in the broker code. That may be an issue if it is called from a program that thinks that this module is still active after its call. Segmentation faults usually occur when Globus functions are called in a non-activated module. For more information see:

http://www.globus.org/common/activation/functions.html

5.1.4 Callbacks

A callback is a C function provided as a parameter to an asynchronous Globus function that is invoked after the call of the function. The Globus call is non-blocking, in that it does not wait for the operation to complete. Instead the Globus call returns immediately. The callback will be invoked after the call in a different thread. Consequently, a synchronization mechanism needs to be used between the callback thread and the program thread that called the asynchronous call so that the program knows when the callback has been made.

To ensure thread safety for the application, a mutex coupled with a condition variable must be used for synchronization. The condition variable is used to send a signal from the callback function to the main program, and the mutex is used

jointly with the condition variable to avoid deadlocks between the waiting thread and the active thread:

- The thread in the main program that calls the asynchronous globus function must call the following globus_thread functions to wait for the completion of the operation:

```
globus_mutex_lock(&mutex);
while(done==GLOBUS_FALSE)
   globus_cond_wait(&cond, &mutex);
globus_mutex_unlock(&mutex);
```

  In this code, done is a boolean variable initialized to false or GLOBUS_FALSE. `done` indicates the state of the operation.

- The callback function must call:

```
globus_mutex_lock(&mutex);
done = GLOBUS_TRUE;
globus_cond_signal(&cond);
globus_mutex_unlock(&mutex);
```

  This mechanism is implemented in this publication via the ITSO_CB class that embeds the done variable as an attribute. ITSO_CB (“ITSO_CB” on page 315) provides the necessary methods:

  - Wait() waits for the completion of the operation.
  - setDone() sets the status of the operation to “done”. The done attribute is actually set to true.
  - IsDone() retrieves the state of the operation (done or not) and checks the value of the done attribute.
  - Continue() resets the value of the attribute `done` to `false`.

Note: In the C/C++ publication examples, an ITSO_CB object, as well as a C callback function that will call its setDone() method, must be provided to the asynchronous globus functions. They take the ITSO_CB object as well as the callback function pointer as arguments. The C function is declared as static, and must be reentrant and is only used to call the ITSO_CB object methods.

5.2 Submitting a job

Before showing programming examples, let us briefly review the options that are available when a job is submitted. The easiest way to do this is to look at the commands that are available. Once you understand the types of things you might do from the command line, it will help you better understand what you must do programmatically when writing your application.

5.2.1 Shells commands

The Globus Toolkit provides several shell commands that can be easily invoked by an application. In this case, the application may be a wrapper script that launches one or more jobs. The commands that can be used to launch a job include:

- **globus-job-run**
- **globus-job-submit**
- **globusrun**
- **gsissh** (not really a Globus job submission command, but provides a secure shell capability using the Globus GSI infrastructure)

All these functions use the Grid Security Infrastructure. Therefore, it is mandatory to always create a valid proxy before running these commands. The proxy can be created with the **grid-proxy-init** command.

globus-job-run

globus-job-run is the simplest way to run a job. The syntax is:

```
globus-job-run <hostname> <program> <arguments>
```

The program must refer to the absolute path of the program. However, by using the -s option, globus will automatically transfer the program to the host where it will be executed:

Example 5-3 globus-job-run example

```
[globus@m0 globus]$ echo "echo Hello World" > MyProg;chmod +x MyProg
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=m0user
Enter GRID pass phrase for this identity:
Creating proxy ................................. Done
Your proxy is valid until: Tue Mar 18 05:23:49 2003
[globus@m0 globus]$ globus-job-run t1 MyProg
GRAM Job failed because the executable does not exist (error code 5)
[globus@m0 globus]$ globus-job-run t1 -s MyProg
Hello World
```

The -: delimiter can be used to submit a multi-request query, as shown in Example 5-4.

Example 5-4 multi-request query

```
globus@m0 globus]$ echo 'echo Hello $1 from $HOSTNAME' > MyProg;chmod +x MyProg
[globus@m0 globus]$ globus-job-run -args You -: a1 -s MyProg -: b1 -s MyProg -:
c1 -s MyProg
```

```
Hello You from a1.itso-apache.com
Hello You from b1.itso-bororos.com
Hello You from c1.itso-cherokee.com
```

globus-job-submit

This shell command submits a job in the background so that you can submit a job, log out of the system, and collect the results later. The job is managed via a URL, also known as a job contact, created at job submission.

The syntax is the same as for globus-job-run except that the program must refer to an absolute path and the -s option cannot be used.

Example 5-5 globus-job-submit example

```
[globus@m0 globus]$ globus-job-submit a1 /myjobs/LongRunningJob
https://a1.itso-apache.com:47573/22041/1047929562/
```

The job contact returned (the `https...` string in the example) can then be used with the following commands:

- **globus-job-status <job contact>** to get the status of the job (pending, active,done, failed, others)
- **globus-job-get-ouput <job contact>** to retrieve the output of the job
- **globus-job-cancel <job contact>** to cancel the job
- **globus-job-clear <job contact>** to clear the files produced by a job

Example 5-6 Retrieving information about a job

```
[globus@m0 globus]$ globus-job-status
https://a1.itso-apache.com:47573/22041/1047929562/
ACTIVE
[globus@m0 globus]$ globus-job-cancel
https://a1.itso-apache.com:47573/22041/1047929562/
Are you sure you want to cancel the job now (Y/N) ?
Y
Job canceled.
NOTE: You still need to clean files associated with the
job by running globus-job-clean <jobID>

[globus@m0 globus]$ globus-job-clean
https://a1.itso-apache.com:47573/22041/1047929562/

    WARNING: Cleaning a job means:
        - Kill the job if it still running, and
        - Remove the cached output on the remote resource
```

```
    Are you sure you want to cleanup the job now (Y/N) ?
Y
Cleanup successful.
```

5.2.2 globusrun

All jobs in the Globus Toolkit 2.2 are submitted by using the RSL language. The RSL language is described in 2.1.2, “Resource management” on page 17. **globusrun** permits you to execute an RSL script.

The -s options starts up a GASS server that can be referenced in the RSL string with the GLOBUSRUN_GASS_URL environment variable. This local GASS server allows the data movement between the compute nodes and the submission node where the **globusrun** command is issued.

The syntax for the **globusrun** command is:

```
globusrun -s -r <hostname> -f <RSL script file>
globusrun -s -r <hostname> ‘RSL script’
```

There is also a -b option (for batch mode) that makes the command return a job contact URL that can be used with:

- **globusrun -status <job contact>** to check the status of a job
- **globusrun -kill <job contact>** to kill a job

Example 5-7 globusrun example

```
[globus@m0 globus]$ echo 'echo Hello $1 from $HOSTNAME' > MyProg;chmod +x
MyProg
[globus@m0 globus]$ globusrun -s -r a1
'&(executable=$(GLOBUSRUN_GASS_URL)'"$PWD"'/MyProg)(arguments=World)'
Hello World from a1.itso-apache.com
```

globus-job-run and **globus-job-submit** actually generate and execute RSL scripts. By using the -dumprsl option, you can see the RSL that is generated and used.

Example 5-8 globus-job-submit -dumprsl example

```
[globus@m0 globus]$ globus-job-submit -dumprsl a1 /bin/sleep 60
  &(executable="/bin/sleep")
   (arguments= "60")
   (stdout=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stdout anExtraTag)
   (stderr=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stderr anExtraTag)
```

5.2.3 GSIssh

GSI-OpenSSH is a modified version of the OpenSSH client and server that adds support for GSI authentication. GSIssh can be used to remotely create a shell on a remote system to run shell scripts or to interactively issue shell commands, and it also permits the transfer of files between systems without being prompted for a password and a user ID. Nevertheless, a valid proxy must be created by using the **`grid-proxy-init`** command.

The problem of unknown sshd host keys is handled as part of the GSIssh protocol by hashing the sshd host key, signing the result with the GSI host certificate on the sshd host, and sending this to the client. With this information the client now has the means to verify that a host key belongs to the host it is connecting to and detect an attacker in the middle.

The Grid Portal Development Kit (GPDK) provides a Java Bean that provides GSIssh protocol facilities to a Java application used in a Web portal. For more information see:

`http://doesciencegrid.org/projects/GPDK/`

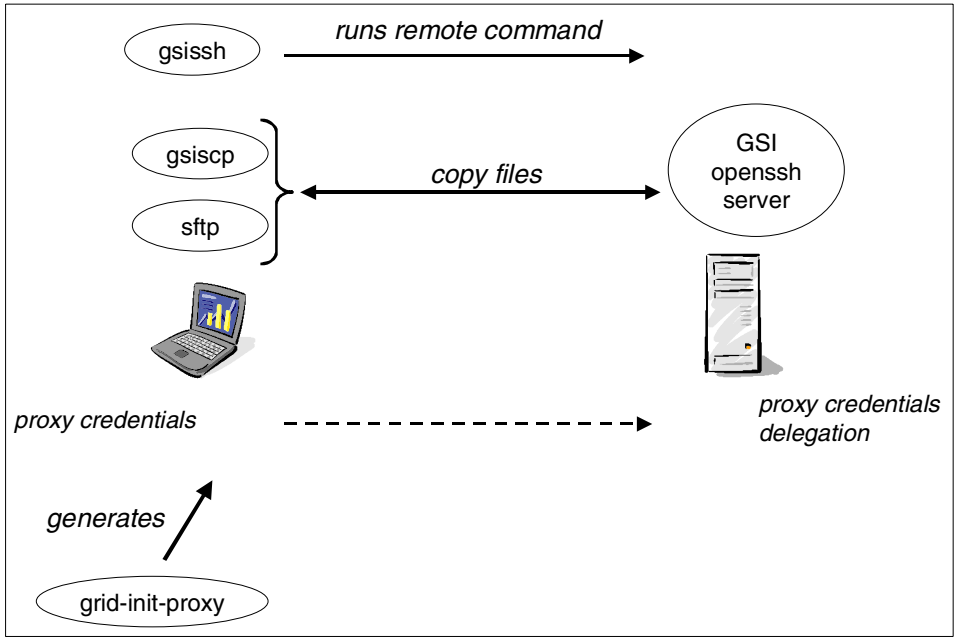


Figure 5-1 GSI-enabled OpenSSH architecture

The installation procedure as well as a complete example is provided in “GSIssh installation” on page 116.

gsissh is used the same way as **ssh**. It cannot use Globus URLs; consequently, files must be staged in and out using **gsiscp** or **sftp**. The executable must be present on the remote host before execution. Below are a few examples.

Example 5-9 gsissh example

```
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=m0user
Enter GRID pass phrase for this identity:
Creating proxy ........................................
.......... Done
Your proxy is valid until: Tue Mar 18 04:33:21 2003
[globus@m0 globus]$ gsissh t1 "date;hostname"
Mon Mar 17 10:33:33 CST 2003
t1.itso-tupi.com
```

The **gsissh** command also embeds and secures the X11 protocol that allows the user to remotely run an application that will be displayed on the local X server. This example runs the Linux monitoring software gkrellm on t1 but will display the graphical interface on m0.

Example 5-10 Running a graphical application through gsissh

```
[globus@m0 globus]$gsissh t1 gkrellm
```

gsissh also supports proxy delegation. That means that once the GSI credentials are created on one node, a user can log onto other nodes and, from there, submit jobs that will use the same GSI credentials. In Example 5-11, a user connects to t1 and from there can submit a job without the need to regenerate a new Globus proxy.

Example 5-11 Proxy delegation support

```
on m0:
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=m0user
Enter GRID pass phrase for this identity:
Creating proxy ........................................
.......... Done
Your proxy is valid until: Tue Mar 18 04:33:21 2003
[globus@m0 globus]$ gsissh t1.itso-tupi.com
Last login: Fri Mar 14 15:16:59 2003 from m0.itso-maya.com

on t1:
[globus@t1 globus]$ globus-job-run a1 -s /bin/hostname
a1.itso-apache.com
[globus@t1 globus]$ grid-proxy-info
subject  : /O=Grid/O=Globus/OU=itso-maya.com/CN=m0user/CN=proxy/CN=proxy
```

```
issuer   : /O=Grid/O=Globus/OU=itso-maya.com/CN=mOuser/CN=proxy
type     : full
strength : 512 bits
timeleft : 11:19:34
```

For more information, see the followings links:

http://www.OpenSSH http://www.openssh.org
http://www.GSIopenssh http://www.nsf-middleware.org/NMIR2/

GSIssh installation

GSIssh middleware is developed by the National Science Foundation Initiative and is not included in the Globus Toolkit. Therefore, it needs to be installed on top of Globus Toolkit 2.2 and its installation requires the Globus Packaging Technology (GPT).

It can be downloaded at the following site:

http://www.nsf-middleware.org/NMIR2/download.asp#GCSS

The installation instructions can be found at

http://www.nsf-middleware.org/documentation/NMI-R2/0/All/allserver_install.htm

GSIssh can be either installed by using a binary bundle (already compiled) or by using a source bundle (that needs to be compiled on site). The installation procedure is very well explained on the NMI Web site (see above).

The following steps summarize the installation procedure for GSIssh using the source package in the case where the Globus Toolkit 2 has been already installed.

1. Download the GSIssh package from the NMI Web site.
2. Set up your environment according your Globus Toolkit environment:

   ```
   export GPT_LOCATION=/usr/local/globus
   export GLOBUS_LOCATION=/usr/local/globus
   ```

3. Build the bundle using GPT's build command.

   ```
   $GPT_LOCATION/sbin/gpt-build -static gsi_openssh-NMI-2.1-src_bundle.tar.gz
   gcc32
   ```

4. Run any post-install setup scripts that require execution.

   ```
   $GPT_LOCATION/sbin/gpt-postinstall
   ```

5. Use GPT's verify command to verify that all of the files were installed properly.

   ```
   $GPT_LOCATION/sbin/gpt-verify
   ```

6. Install gsissh as a service.

```
cp /usr/local/globus/sbin/SXXsshd /etc/rc.d/init.d/gsissh
chkconfig --level 3 gsissh on
service gsissh start
```

Note: GSIssh can be installed concurrently with a non-gsi ssh server. However, since they both default to using the same port, you have to modify the port on which the GSIssh will listen for requests. To do this, edit /etc/rc.d/init.d/gsissh and assign a value to SSHD_ARGS, for example, SSHD_ARGS="-p 24", to listen on port 24.

You will then need to specify this port for all **`gsissh`**, **`gsiscp`**,and **`gsisftp`** commands:

```
gsissh -p 24 g3.itso-guarani.com hostname
```

5.2.4 Job submission skeleton for C/C++ applications

To submit a job in a C or C++ program, an RSL string describing the job must be provided. The globus_gram_client API provides an easy API for job submission. Two kinds of functions can be used:

- Blocking calls that wait for the completion of the jobs before returning
- Non-blocking or asynchronous calls that return immediately and call a “callback” function when the operation has completed or to inform the main program about the status of the asynchronous operation

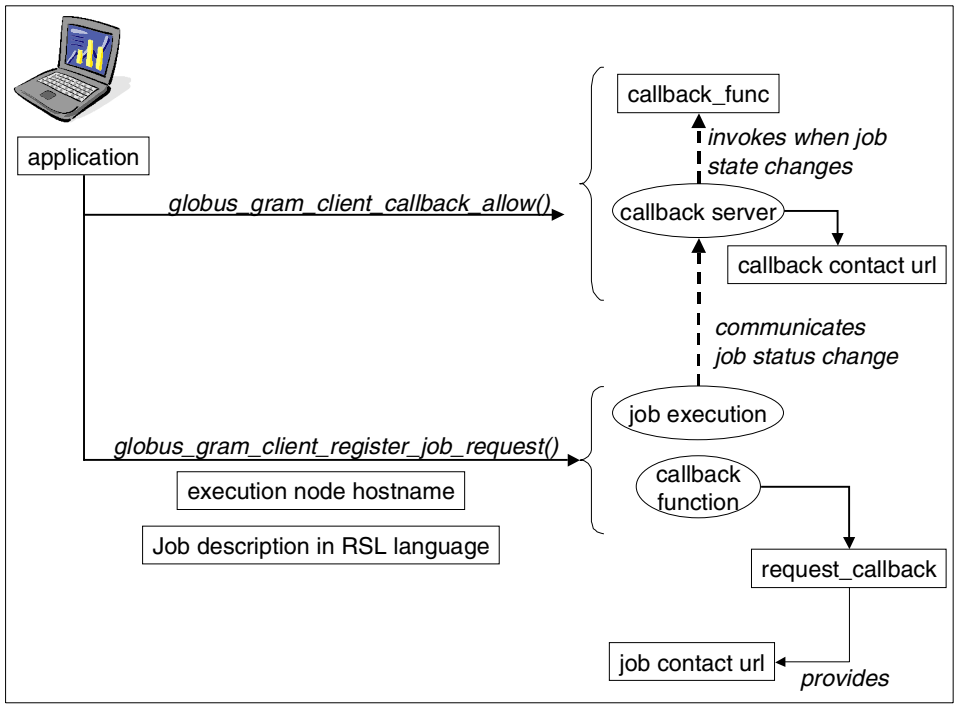


Figure 5-2 Job submission using non-blocking calls

Note: The documentation of the globus_gram_client API is available at:

`http://www-unix.globus.org/api/c/globus_gram_client/html/index.html`

We only cover non-blocking calls in this chapter, as they are the more complicated from a programming perspective, but often more desirable from an application perspective. Non-blocking calls allow the application to submit several jobs in parallel rather than wait for one job to finish before submitting the next.

Job submission

The ITSO_GRAM_JOB class provided in “itso_gram_job.C” on page 321 provides an asynchronous implementation in C++ of a job submission. It is derived from ITSO_CB. ITSO_GRAM_JOB wraps C Globus GRAM API functions in its methods. Its implementation is based on the C example available in “Submitting a job” on page 358.

The first step is to create the GRAM server on the execution node that will monitor the status of the job and associate a callback with this job. This is

achieved by calling the function globus_gram_client_callback_allow(). In the Submit() method of the class ITSO_GRAM_JOB, we find:

```
globus_gram_client_callback_allow(itso_gram_job::callback_func,
                      (void *) this,
                      &callback_contact);
```

The ITSO_GRAM_JOB object, derived from ITSO_CB, is itself passed as an argument so that the callback could invoke the method of this object via the 'this' pointer. It is associated, as well as the callback_function, with globus_gram_client_callback_allow()to manage its asynchronous behavior. &callback_contact is the job contact URL that will be set after this call. The setDone(), setFailed() methods of the ITSO_GRAM_JOB object (implemented in ITSO_CB) will permit the callback to modify the status of the job in the application. Note that the status of the job in the application is independently managed from the status of the job that is be obtained via the following globus calls:

```
globus_gram_client_job_status() (blocking call)
globus_gram_client_resgister_job_status() (non-blocking call)
```

Here is an example of a callback to the globus_gram_client_callback_allow() function. Note that callbacks have a well-defined prototype that depends on the Globus functions they are associated with. The job contact URL is received as an argument as well as the ITSO_GRAM_JOB object pointer.

Example 5-12 globus_gram_client_callback_allow() callback function

```
static void callback_func(void * user_callback_arg,
                  char * job_contact,
                  int state,
                  int errorcode)
{
   //The ITSO_GRAM_JOB object is retrieved in the callback via the first
   //argument that allows to pass any kind of pointer to the callback.
   //This is the second argument of the globus_gram_client_callback_allow()
   //function
    ITSO_GRAM_JOB*  Monitor = (ITSO_GRAM_JOB*) user_callback_arg;

    switch(state)
    {
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
        cout << "Staging file in on: " << job_contact << endl;
          break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
          cout << "Staging file out  on: " << job_contact << endl;
          break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
          break; /* Reports state change to the user */
```

```
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
          cerr << "Job Failed on: " << job_contact << endl;
          Monitor->SetFailed();
          Monitor->setDone();
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
          cout << "Job Finished on: " << job_contact << endl;
          Monitor->setDone();
          break; /* Reports state change to the user */
    }
}
```

The next step is to submit the job itself. This is achieved by calling the globus_gram_client_register_job_request() function that is an asynchronous or non-blocking call, that also needs (in our example) a C callback function and an ITSO_CB object. The request_cb attribute of the class ITSO_GRAM_JOB will be used for this purpose. The callback function used with globus_gram_client_register_job_request() is request_callback(). See “ITSO_GRAM_JOB” on page 316 for implementation details. It calls the method SetRequestDone() of the ITSO_GRAM_JOB object that itself calls the setDone() method of the ITSO_CB class through the request_cb attribute.

The RSL submission string is passed as an argument, as well as the host name of the execution node, to globus_gram_client_register_job_request(). GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL specifies that we want to monitor all states (done, failed, staging files). The ITSO_GRAM_JOB object itself is passed as an argument ((void*) this). This way the callback can invoke its SetRequestDone() method. See Example 5-14 on page 121.

Example 5-13 globus_gram_client_register_job_request call

```
int rc = globus_gram_client_register_job_request(res.c_str(),
                         rsl.c_str(),
                         GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
                         callback_contact,
                         GLOBUS_GRAM_CLIENT_NO_ATTR,
                         itso_gram_job::request_callback,
                         (void*) this);
```

Here is an example of a globus_gram_client_register_job_request() callback. The callback is called whether the job has been submitted successfully or not.

Example 5-14 globus_gram_client_register_job_request() callback

```
static void request_callback(void * user_callback_arg,
                             globus_gram_protocol_error_t failure_code,
                             const char * job_contact,
                             globus_gram_protocol_job_state_t state,
                             globus_gram_protocol_error_t errorcode) {
    ITSO_GRAM_JOB*  Request = (ITSO_GRAM_JOB*) user_callback_arg;
    cout << "Contact on the server " << job_contact << endl;
    Request->SetRequestDone(job_contact);
}
```

The callback calls the `SetRequestDone()` method of the `ITSO_GRAM_JOB` object that actually calls the setDone() method of the request_cb ITSO_CB object associated with the function globus_gram_client_register_job_request().

The Submit() method of the ITSO_GRAM_JOB class implements the job submission.

Example 5-15 GRAM job submission via an ITSO_GRAM_JOB object

```
bool ITSO_GRAM_JOB::Submit(string res, string rsl) {
    failed=false;
    globus_gram_client_callback_allow(itso_gram_job::callback_func,
                      (void *) this,
                      &callback_contact);
    int rc = globus_gram_client_register_job_request(
                      res.c_str(),
                      rsl.c_str(),
                      GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
                      callback_contact,
                      GLOBUS_GRAM_CLIENT_NO_ATTR,
                      itso_gram_job::request_callback,
                      (void*) this);
    if (rc != 0) /* if there is an error */
    {
        printf("TEST: gram error: %d - %s\n",
                rc,
                /* translate the error into english */
                globus_gram_client_error_string(rc));
        return true;
    }
   else
      return false;
};
```

Checking if we can submit a job on a node

The function globus_gram_client_ping() can be used for diagnostic purposes to check whether a host is available to run the job.

Example 5-16 CheckHost.C

```
#include "globus_gram_client.h"
#include <iostream>

int main(int argc, char **   argv)
{

    globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);

    cout << argv[1];
    if (!globus_gram_client_ping(argv[1]))
           cout << " is okay " << endl;
    else
           cout << " cannot be used " << endl;

    globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
}
```

To compile the above program:

1. Generate the globus variables used in the Makefile.

   ```
   globus-makefile-header --flavor gcc32 globus_gram_job > globus_header
   ```

2. Then use the following Makefile.

   ```
   include globus_header
   all: CheckNodes

   %.o: %.C
      g++ -g -c -I. $(GLOBUS_CPPFLAGS) $< -o $@

   CheckNodes: CheckNodes.o
      g++ -g -o $@ $(GLOBUS_CPPFLAGS)  $(GLOBUS_LDFLAGS) $^
   $(GLOBUS_PKG_LIBS)
   ```

3. Issue **make** to compile.

When this program executes, you will see results similar to the following:

```
[globus@m0 JYCode]$ ./CheckNodes a1.itso-tupi.com
a1.itso-tupi.com cannot be used
[globus@m0 JYCode]$ ./CheckNodes t1.itso-tupi.com
t1.itso-tupi.com is okay
```

Job resubmission

In this example, by using ITSO_GRAM_JOB , we submit a job, check if it has failed, and, if so, submit it again to another host.

One (simple) method is to get three nodes from the broker and submit the job to the next node when it fails on the previous one.

The job state management is managed in the callback function shown in Example 5-12 on page 119. We declare that we want to monitor all changes in the state of the job (GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL option passed to the globus_gram_client_register_job_request() function). Then the callback modifies (or not) the status of the job via the SetFailed() method provided by the ITSO_GRAM_JOB class.

The SureJob.C program is the implementation of such a job submission that checks the state of the job after the Wait() method has returned, by using the HasFailed() method. If failed, the job is submitted to the next host provided by the broker.

HasFailed() simply checks the value of a boolean attribute of an ITSO_GRAM_JOB object that becomes true when the job has failed. This attribute is set to false by default, but can be modified in the callback function of the globus_gram_client_callback_allow() function by calling the setFailed() method of the ITSO_GRAM_JOB object when a failure is detected.

The broker returns a vector of hostnames via the GetLinuxNodes() call (see “Broker example” on page 127 for more details). It internally tests if the user is able to submit a job on the node with a globus ping before returning the vector of host names. For various reasons the job may fail to execute on this node, and SureJob.C provides a simple way to overcome this failure.

Example 5-17 SureJob.C

```
#include <string>
#include <vector>
#include <broker.h>
#include "globus_gram_client.h"
#include "itso_gram_job.h"

using namespace itso_broker;

int main(int argc, char **   argv)
{
    vector<string> Nodes;
    GetLinuxNodes(Nodes,3);

    // Quickly check if we can run a job
```

```
    globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);

    ITSO_GRAM_JOB job;
    vector<string>::iterator i;
    for(i=Nodes.begin();i!=Nodes.end();++i) {
      cout << "Try to submit on " << *i << endl;
      job.Submit(*i,"&(executable=/bin/hostname)");
      job.Wait();
      if (!job.HasFailed())
         break;
    };

    globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
}
```

Here is the result when a1 and c2 are down.

```
[globus@m0 JYCode]$ ./SureJob
Try to submit on a1.itso-apache.com
Contact on the server https://a1.itso-apache.com:48181/27222/1047945694/
Job Failed on: https://a1.itso-apache.com:48181/27222/1047945694/
Try to submit on c2.itso-cherokee.com
Contact on the server https://c2.itso-cherokee.com:40304/20728/1047945691/
Job Failed on: https://c2.itso-cherokee.com:40304/20728/1047945691/
Try to submit on c1.itso-cherokee.com
Contact on the server https://c1.itso-cherokee.com:47993/25310/1047945698/
Job Finished on: https://c1.itso-cherokee.com:47993/25310/1047945698/
```

5.2.5 Simple broker

A user application should not have to care about locating the resources it needs. It just needs to describe to a broker the kind of resources it will use to run the applications: Operating systems, SMP, number of nodes, available applications, available storage, and so on. This task needs to be done at the application level via a component called a broker that can be implemented in the application itself, or as a service that will be queried by the applications. The Globus Toolkit 2.2 does not provide a broker implementation, but it does provide the necessary functions and framework to create one through the MDS component.

The broker software will communicate via the LDAP protocol in the Globus Toolkit 2 with the GIIS and GRIS servers. The broker can be linked with other information stored in databases or plain files that provide other information such as customer service level agreement, resources topology, network problems, and cost of service. This third-party data may influence the decisions of what resource to use in conjunction with the technical information provided by default with MDS.

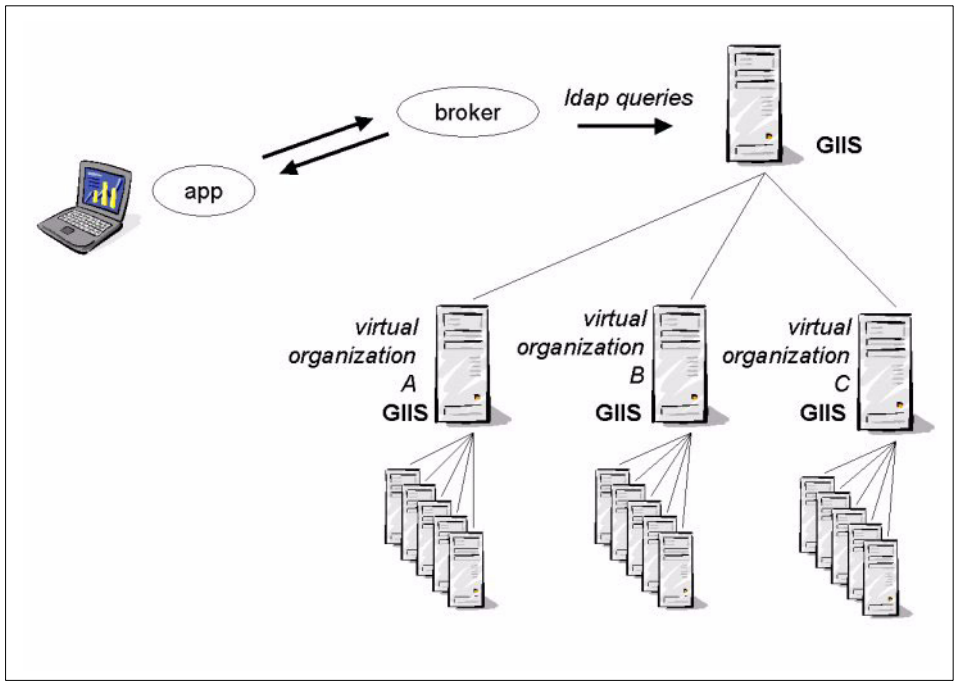


Figure 5-3 Working with a broker

Using Globus Toolkit tools

`grid-info-search` as well as **`ldapsearch`** are the shell tools used to query information through the GIIS server. The -h option allows the user to specify a specific host, usually the master GIIS server (on top in Figure 5-3), m0 in our lab environment. The connection to the GIIS can be controlled through GSI security, such that a valid proxy certificate needs to be generated before running either of the two commands:

```
d1user@d1 d1user]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-dakota.com/CN=d1user
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Sat Mar 15 06:55:55 2003
```

An LDAP query implements sophisticated query operations that include:

- Logic operators: AND (`&`), OR (`|`), and NOT (`!`)
- Value operators: `=`, `>=`, `<=`, `-=` (for approximate matching)

For example, here is a way to look up host names of the resources of all nodes running Linux that use a Pentium processor with a CPU speed greater than 500 Mhz:

```
ldapsearch -x -p 2135 -h m0 -b "mds-vo-name=maya,o=grid" -s sub
'(&(Mds-Os-name=Linux)(Mds-Cpu-model=Pentium *)(Mds-Cpu-speedMHz>=500))'
Mds-Host-hn
version: 2

#
# filter: (&(Mds-Os-name=Linux)(Mds-Cpu-model=Pentium
II*)(Mds-Cpu-speedMHz>=500))
# requesting: Mds-Host-hn
#

# a1.itso-apache.com, apache, maya, Grid
dn: Mds-Host-hn=a1.itso-apache.com,Mds-Vo-name=apache,Mds-Vo-name=maya,o=Grid
Mds-Host-hn: a1.itso-apache.com

# t2.itso-tupi.com, tupi, maya, Grid
dn: Mds-Host-hn=t2.itso-tupi.com,Mds-Vo-name=tupi,Mds-Vo-name=maya,o=Grid
Mds-Host-hn: t2.itso-tupi.com

# t1.itso-tupi.com, tupi, maya, Grid
dn: Mds-Host-hn=t1.itso-tupi.com,Mds-Vo-name=tupi,Mds-Vo-name=maya,o=Grid
Mds-Host-hn: t1.itso-tupi.com
```

The following command can be included in a program to retrieve the list of the machines that match the criteria:

```
[d1user@d1 d1user]$ ldapsearch -x -p 2135 -h m0 -b "mds-vo-name=maya,o=grid" -s
sub '(&(Mds-Os-name=Linux)(Mds-Cpu-model=Pentium *)(Mds-Cpu-speedMHz>=500))'
Mds-Host-hn | awk '/Mds-Host-hn:/ { print $2 }' | xargs

t2.itso-tupi.com t1.itso-tupi.com a1.itso-apache.com
```

In the next example, we look for all machines that have a Pentium processor and that either runs at a frequency greater than 500 Mhz, or has more than 5 Gb of available diskspace.

```
ldapsearch -x -p 2135 -h m0 -b "mds-vo-name=maya,o=grid" -s sub
'(&(Mds-Os-name=Linux)(Mds-Cpu-model=Pentium*)(|(Mds-Cpu-speedMHz>=500)(Mds-Fs-
Total-sizeMB>=5000)))' Mds-Host-hn | awk '/Mds-Host-hn:/ { print $2 }' | xargs

a1.itso-apache.com a2.itso-apache.com b2.itso-bororos.com d2.itso-dakota.com
d1.itso-dakota.com t2.itso-tupi.com t3.itso-tupi.com t1.itso-tupi.com
t0.itso-tupi.com c2.itso-cherokee.com c1.itso-cherokee.com
```

Graphical tools

There are a variety of GUI tools can be used to browse the Globus MDS server. Under Linux, a graphical client named gq permits easy browsing. If not available on your distribution, it can be downloaded from the following URL:

`http://biot.com/gq/`

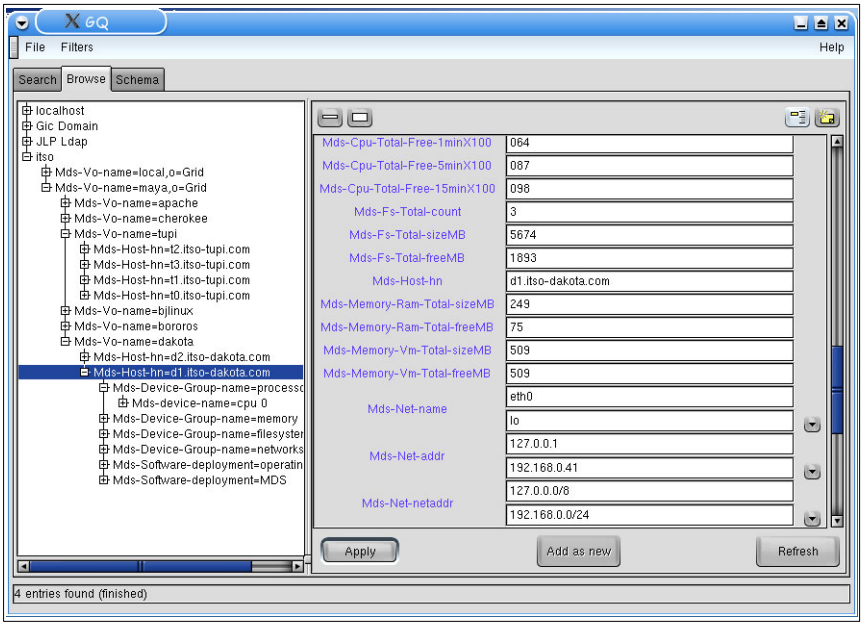


Figure 5-4 GQ LDAP browser

Broker example

In our example, we use a basic broker that can be called via a function that takes the number of required Linux nodes as a parameter and a vector of strings (as defined in C++) that will contain the list of nodes when the function returns.

This simple broker checks the average CPU workload measured in a fifteen-minute period of time, the number or processors, and the CPU speed. All this information is available from the GIIS server for each host as Mds-Cpu-Free-15mnX100, Mds-Cpu-Total-count, and Mds-Cpu-speedMHz attributes, respectively. The broker multiplies the three attributes and performs a quick sort to return the nodes that apparently are the best available. Each node is checked with the function globus_gram_client_ping() to check if the node is available.

The complete source code is available in “Broker.C” on page 327.

We use the LDAP API provided by the Globus Toolkit 2.2 to send the request to the main GIIS server located on m0 in our lab environment. The definition is statically defined in the program, but can be easily provided as a parameter to the GetLinuxNodes() function if needed:

```
#define GRID_INFO_HOST "m0"
#define GRID_INFO_PORT "2135"
#define GRID_INFO_BASEDN "mds-vo-name=maya, o=grid"
```

In the function `GetLinuxNodes()`, the connection with MDS is managed by a structure of type LDAP* initialized by the two calls, ldap_open() and ldap_simple_bind_s(), for the connection.

Example 5-18 LDAP connection

```
    char *           server   = GRID_INFO_HOST;
    int              port     = atoi(GRID_INFO_PORT);
    char *           base_dn  = GRID_INFO_BASEDN;
    LDAP *           ldap_server;
   /* Open connection to LDAP server */
    if ((ldap_server = ldap_open(server, port)) == GLOBUS_NULL)
    {
        ldap_perror(ldap_server, "ldap_open");
        exit(1);
    }

    /* Bind to LDAP server */
    if (ldap_simple_bind_s(ldap_server, "", "") != LDAP_SUCCESS)
    {
        ldap_perror(ldap_server, "ldap_simple_bind_s");
        ldap_unbind(ldap_server);
        exit(1);
    }
```

We are only interested in the resources running the Linux operating system. This can be expressed by the following LDAP query:

```
(&(Mds-Os-name=Linux)(Mds-Host-hn=*))
```

Then we can submit the query, as shown in Example 5-14 on page 121.

Example 5-19 Submitting the LDAP query

```
string filter= "(&(Mds-Os-name=Linux)(Mds-Host-hn=*))";
if (ldap_search_s(ldap_server, base_dn,
                      LDAP_SCOPE_SUBTREE,
                      const_cast<char*>(filter.c_str()), attrs, 0,
                      &reply) != LDAP_SUCCESS)
    {
        ldap_perror(ldap_server, "ldap_search");
```

```
        ldap_unbind(ldap_server);
        exit(1);
    }
```

The result of the query is a set of entries that match the query. Each entry is itself a set of attributes and their values. The ldap_first_entry() and ldap_next_entry() functions allow us to walk the list of entries. ldap_first_attribute() and ldap_next_attribute() allow us to walk the attribute list, and ldap_get_values() is used to return their value.

Example 5-20 Retrieving results from Globus MDS

```
LDAPMessage *   reply;
LDAPMessage *   entry;
vector<Host*>   nodes;

for (entry = ldap_first_entry(ldap_server, reply);
entry != GLOBUS_NULL;
entry = ldap_next_entry(ldap_server, entry) )
{
   //cout << endl << ldap_get_dn( ldap_server, entry ) << endl;
   BerElement * ber;
   char**       values;
   char *       attr;
   char *       answer = GLOBUS_NULL;
   string hostname;
   int cpu;
   for (attr = ldap_first_attribute(ldap_server,entry,&ber);
      attr != NULL;
      attr = ldap_next_attribute(ldap_server,entry,ber) )
      {

         values = ldap_get_values(ldap_server, entry, attr);
         answer = strdup(values[0]);
         ldap_value_free(values);
         if (strcmp("Mds-Host-hn",attr)==0)
            hostname=answer;
         if (strcmp("Mds-Cpu-Free-15minX100",attr)==0)
            cpu=atoi(answer);
         if (strcmp("Mds-Cpu-Total-count",attr)==0)
            cpu_nb=atoi(answer);
         if (strcmp("Mds-Cpu-speedMHz",attr)==0)
            speed=atoi(answer);
         //printf("%s %s\n", attr, answer);

   }
// check if we can really use this node
if (!globus_gram_client_ping(hostname.c_str()))
```

```
      nodes.push_back(new Host(hostname,speed*cpu_nb*cpu/100));
   };
```

Only valid nodes (that are available) are selected. The globus_gram_client_ping() function from the globus_gram_client API is used for this purpose. We also calculate a weight for each node, speed*cpu_nb*cpu/100. The higher the weight is, the higher our ranking of the node will be. The broker will return the best nodes first, as shown in Example 5-21.

Example 5-21 Check the host

```
   if (!globus_gram_client_ping(hostname.c_str()))
      nodes.push_back(new Host(hostname,speed*cpu_nb*cpu/100));
```

In a real environment, the broker should take into account a variety of factors and information. Not all of the information has to come from MDS. For instance, some other factors that might affect the broker's choice of resources could be:

- Service level agreements
- Time range of utilization
- Client location
- And many others

The broker finally proceeds to sort and set up the vector of strings that will be returned to the calling function. This logic, as well as the LDAP query, can be easily customized to meet any specific requirements, as shown in Example 5-22.

Example 5-22 Broker algorithm implementation

```
class Host {
      string    hostname;
      long      cpu;
   public:
      Host(string h,int c) : hostname(h), cpu(c) {};
      ~Host() { };
      string getHostname() { return hostname; };
      int getCpu() { return cpu; };
};


bool predica(Host* a, Host* b) {
 return (a->getCpu() > b->getCpu());
}
..............................
   globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
..............................
   { // for each entry do
      values = ldap_get_values(ldap_server, entry, attr);
```

```
            answer = strdup(values[0]);
            ldap_value_free(values);
         if (strcmp("Mds-Host-hn",attr)==0)
            hostname=answer;
         if (strcmp("Mds-Cpu-Free-15minX100",attr)==0)
             cpu=atoi(answer);
         if (strcmp("Mds-Cpu-Total-count",attr)==0)
             cpu_nb=atoi(answer);
         if (strcmp("Mds-Cpu-speedMHz",attr)==0)
             speed=atoi(answer);
         //printf("%s %s\n", attr, answer);
      }
      // check if we can really use this node
      if (!globus_gram_client_ping(hostname.c_str()))
         nodes.push_back(new Host(hostname,speed*cpu_nb*cpu/100));

       };
      sort(nodes.begin(),nodes.end(),predica);
       vector<Host*>::iterator i;
       for(i=nodes.begin();(n>0) && (i!=nodes.end());n--,i++){
          res.push_back((*i)->getHostname());
          //cout << (*i)->getHostname() << " " << (*i)->getCpu() << endl;
          delete *i;
       }
       for(;i!=nodes.end();++i)
          delete *i;

globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
```

Example 5-23 is a quick example that uses the broker.C implementation. The application takes the first argument as the number of required nodes running the Linux operating system.

Example 5-23 Application using GetLinuxNodes() to get n nodes

```
#include <string>
#include <vector>
#include <broker.h>

using namespace itso_broker;

int main(int argc, char **   argv)
{
    vector<string> Y;
    GetLinuxNodes(Y,atoi(argv[1]));
    vector<string>::iterator i;
    for(i=Y.begin();i!=Y.end();++i)
            cout << *i << endl;
```

```
}
```

Executing the program in our environment results in:

```
[globus@m0 GLOBUS]$ ./mds 6
c1.itso-cherokee.com
d2.itso-dakota.com
a1.itso-apache.com
t1.itso-tupi.com
c2.itso-cherokee.com
d1.itso-dakota.com
```

Note: Do not forget to modify the MDS attributes to suit your environment in broker.C:

```
#define GRID_INFO_HOST "m0"
#define GRID_INFO_PORT "2135"
#define GRID_INFO_BASEDN "mds-vo-name=maya, o=grid"
```

5.3 Summary

In this chapter, we have introduced the C/C++ programming environment for the Globus Toolkit and provided several samples for submitting jobs and searching for resources.

In the next chapter, we will provide samples written in Java that touch most of the components of the Globus Toolkit.

6

Programming examples for Globus using Java

In the previous chapter, some examples of using the Globus Toolkit with C bindings were provided. In this chapter, we provide Java programming examples for most of the services provided by the Globus Toolkit.

Though the Globus Toolkit is shipped with bindings that can be used with C or C++, our examples here are based on Java. We have done this for a few reasons. First, Java is a popular language that many of our readers may be able to read well enough to understand the concepts we are conveying. Second, future versions of the Globus Toolkit will likely ship with Java bindings, and it is likely that more and more application development will utilize Java.

To use Java with the current Globus Toolkit, V2.2.4, you can use the Java Commodity Kit (JavaCoG). More information on CoGs is available at:

`http://www-unix.globus.org/cog/`

Specifically, we recommend *The Java CoG Kit User Manual*, available at:

`http://www.globus.org/cog/manual-user.pdf`

This manual describes the Java CoG toolkit in detail, and describes its installation, configuration, and usage. This chapter assumes that the reader is familiar with the referenced manual.

© Copyright IBM Corp. 2003. All rights reserved.

6.1 CoGs

Commodity Grid Kits is Globus' way to integrate Globus tools into existing platforms. CoG Kits allow users to provide Globus Toolkit functionality within their code without calling scripts, or in some cases without having Globus installed. There are several COGs available for GT2 development. CoGs are currently available for Java, Python, CORBA, Perl, and Matlab.

The Java CoG Kit is the most complete of all the current CoG Kits. It is an extension of the Java libraries and classes that provides Globus Toolkit functionality. The most current version is 1.1 alpha and is compatible with GT2 and GT3. The examples in this chapter use JavaCoG Version 1.1a. JavaCoG provides a pure Java implementation of the Globus features.

This chapter provides examples in Java for interfacing with the following Globus components/functions:

- Proxy: Credential creation and destruction
- GRAM: Job submission and job monitoring
- MDS: Resource searching
- RSL: Resource specification and job execution
- GridFTP: Data management
- GASS: Data management

6.2 GSI/Proxy

The JavaCoG 1.1a Toolkit provides a sample application written in Java to create a proxy. It has the same name as the standard Globus command line function, **`grid-proxy-init`**.

This tool, by default, creates a proxy in a format that will be used in Globus Toolkit V3, which is not compatible with Globus Toolkit V2. To create a proxy valid for Globus Toolkit V2.2, the -old option must be set. This can be done by simply passing -old as a parameter when executing the Java version of **`grid-proxy-init`**.

In order to create a proxy from an application, the JavaCoG Kit must be installed and configured. The proper configuration will provide correct paths to the necessary files in the cog.properties file. The toolkit provides an easy way to read the cog.properties file.

Important: The JavaCoG 1.1a provides support to both GT3 and GT2.2 proxies. By default, it will create a GT3 proxy. In order to create a GT2.2 proxy, the proxyType field must be set properly.

Creating a proxy

This is a basic programming example that shows how to create a proxy compatible with the Globus Toolkit V2.2.

The CoGProperties class provides an easy way to access the cog.properties file, where all file locations are stored.

Example 6-1 Creating a proxy (1 of 3)

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.security.PrivateKey;
import java.security.cert.X509Certificate;

import org.globus.common.CoGProperties;
import org.globus.gsi.CertUtil;
import org.globus.gsi.GSIConstants;
import org.globus.gsi.GlobusCredential;
import org.globus.gsi.OpenSSLKey;
import org.globus.gsi.bc.BouncyCastleCertProcessingFactory;
import org.globus.gsi.bc.BouncyCastleOpenSSLKey;
import org.globus.gsi.proxy.ext.ProxyCertInfo;
import org.globus.util.Util;

/**
 * GridProxy
 *
 * Used to create a proxy
 */
public class GridProxy {

   X509Certificate certificate;
   PrivateKey userKey = null;
   GlobusCredential proxy = null;
   ProxyCertInfo proxyCertInfo = null;
   int bits = 512;
   int lifetime = 3600 * 12;
   int proxyType;

   //Environment Setup
   CoGProperties properties = CoGProperties.getDefault();
```

```
   String proxyFile = properties.getProxyFile();
   String keyFile = properties.getUserKeyFile();
   String certFile = properties.getUserCertFile();
```

The private key is encrypted in the keyfile. Using the OpenSSL libraries, we can load the private key and decrypt by providing a password.

The CA public key can be loaded using the CertUtils class.

Example 6-2 Creating a proxy (2 of 3)

```
public void createProxy() throws Exception {
   System.out.println("Entering createProxy()");

   //loading certificate
   System.out.println("Loading Certificate...");
   certificate = CertUtil.loadCertificate(certFile);

   String dn = certificate.getSubjectDN().getName();
   System.out.println("Your identity: " + dn);

   //loading key
   System.out.println("Loading Key...");
   OpenSSLKey sslkey = new BouncyCastleOpenSSLKey(keyFile);
   if (sslkey.isEncrypted()) {
      String pwd = null;
      pwd = Util.getInput("Enter GRID pass phrase: ");
      sslkey.decrypt(pwd);
   }
   userKey = sslkey.getPrivateKey();
```

In order to create the proxy, we will create a new certificate using our private key. This certificate is marked to be a proxy and it has a lifetime. It is important to note the proxyType variable, which can be used to generate a Globus Toolkit V3 or V2.2 compatible proxy.

Example 6-3 Creating a proxy (3 of 3)

```
//signing
System.out.println("Signing...");

proxyType = GSIConstants.GSI_2_PROXY; //switch here between GT2/GT3

BouncyCastleCertProcessingFactory factory =
   BouncyCastleCertProcessingFactory.getDefault();

proxy =
   factory.createCredential(
```

```
       new X509Certificate[] { certificate },
       userKey,
       bits,
       lifetime,
       proxyType,
       proxyCertInfo);

System.out.println(
   "Your proxy is valid until "
      + proxy.getCertificateChain()[0].getNotAfter());

//file creation
System.out.println("Writing File...");
OutputStream out = null;
try {
   out = new FileOutputStream(proxyFile);

   // write the contents
   proxy.save(out);
} catch (IOException e) {
   System.err.println(
      "Failed to save proxy to a file: " + e.getMessage());
   System.exit(-1);
} finally {
   if (out != null) {
      try {
         out.close();
      } catch (Exception e) {
      }
   }
}

System.out.println("Exiting createProxy()");
}
```

Retrieving credentials from an existing proxy

In order to retrieve the credentials from an existing proxy to be used within the application, the proxy file must be loaded. The proxy file can be located using the cog.properties file or by just specifying the file name.

Example 6-4 Retrieving credentials

```
GlobusCredential gcred = new GlobusCredential("/tmp/x509up_u1101");
cred = new GlobusGSSCredentialImpl(gcred, GSSCredential.DEFAULT_LIFETIME);
```

Destroying the proxy

As the proxy is actually a file, destroying the proxy is quite simple. The file can just be deleted.

Example 6-5 Destroying a proxy

```
public void proxyDestroy() {

   File file = null;
   String proxyfile = CoGProperties.getDefault().getProxyFile();
   if (proxyfile == null)
      return;
   file = new File(proxyfile);

   Util.destroy(file);
}
```

6.3 GRAM

The Java CoG Kit provides two packages to access the GRAM API and run Gram jobs:

- org.globus.gram
- org.globus.gram.internal

The org.globus.gram.internal package contains, as the name indicates, only internal classes that are used by the main org.globus.gram package. The org.globus.gram package provides the GRAM client API.

Inside org.globus.gram the most important basic classes are:

- GramJob - Class
- GramJobListener - Interface
- GramException - Exception

6.3.1 GramJob

This class represents a GRAM job you can submit to a gatekeeper. It also provides methods to cancel the job, register and unregister a callback method, and send signal commands.

6.3.2 GramJobListener

This interface is used to listen for a status change of a GRAM job.

6.3.3 GramException

This class defines the exceptions thrown by a GRAM job.

GRAM example

This example will submit a simple job to a known resource manager. It shows the simplest case, where all you need is an RSL string to execute and the resource manager name. Note that the GRAMTest class implements the GramJobListener interface. This way we get status updates on our job from the resource manager. This example will create a new directory on the server called /home/globus/testdir.

Example 6-6 GRAM example (1 of 2)

```
import org.globus.Gram;
import org.globus.GramJob;
import org.globus.GramJobListener;

/**
 * Basic GRAM example
 * This example submits a simple Gram Job
 */

public class GRAMTest implements GramJobListener {

   //Method called by the listener when the job status changes
   public void statusChanged(GramJob job) {
      System.out.println(
         "Job: "
            + job.getIDAsString()
            + " Status: "
            + job.getStatusAsString());
   }
```

The first thing to do is to create the GRAM job using the RSL string as a parameter. Job status listeners can be attached to the GRAM job to monitor the job. The job is submitted to the resource manager by issuing job.request().

Tip: If you want to check if you are allowed to submit a job to a specific resource manager, the method Gram.ping(rmc) can be issued.

Example 6-7 GRAM example (2 of 2)

```
   private void runJob() {
      //RSL String to be executed
      String rsl =
"&(executable=/bin/mkdir)(arguments=/home/globus/testdir)(count=1)";
```

```
        //Resource Manager Contact
        String rmc = "t2.itso-tupi.com";

        //Instantiating the GramJob with the RSL
        GramJob job = new GramJob(rsl);
        job.addListener(this);

        //Pinging resource contact to check if we are allowed to use it
        try {
           Gram.ping(rmc);
        } catch (Exception e) {
           System.out.println("Ping Failed: " + e.getMessage());
        }

        System.out.println("Requesting Job...");

        try {
           job.request(rmc);

        } catch (Exception e) {
           System.out.println("Error: "+ e.getMessage());
        }

        job.removeListener(this);
     }

     public static void main(String[] args) {

        GRAMTest run = new GRAMTest();
        run.runJob();

        System.out.println("All Done.");
     }
}
```

6.4 MDS

MDS gives users the ability to obtain vital information about the grid and grid resources. It utilizes LDAP to execute queries. Users can retrieve this information by using the **grid-info-search** command line tool. The JavaCog Kit Version 1.1a provides a Java version of **grid-info-search** that does not use MDS. Because the MDS class itself is deprecated, users should use JNDI with LDAP or the Netscape Directory SDK to access MDS functions with the JavaCog.

6.4.1 Example of accessing MDS

Example 6-8 is a condensed version of the GridInfoSearch class provided in the org.globus.tools package for the JavaCog Kit Version 1.1a. The MyGridInfoSearch class uses GSI authentication. It uses JNDI to connect to the LDAP server and searches for the object class specified by a variable. It is important to note that when using this MyGridProxyInit class that you must have a valid Globus proxy and your CLASSPATH must contain the location all of the JavaCog jar files along with the current directory. Without having a valid Globus proxy you will receive the error `Failed to search: GSS-OWNYQ6NTEOAUVGWG`. Without having the proper CLASSPATH you will receive the error `Failed to search: SASL support not available: GSS-OWNYQ6NTEOAUVGWG`.

Example 6-8 shows the import statements and variable declarations for the MyGridInfoSearch class.

Example 6-8 GridInfoSearch example (1 of 4)

```
import java.util.Hashtable;
import java.util.Enumeration;
import java.net.InetAddress;
import java.net.UnknownHostException;

import javax.naming.Context;
import javax.naming.NamingEnumeration;
import javax.naming.NamingException;
import javax.naming.directory.Attribute;
import javax.naming.directory.SearchControls;
import javax.naming.directory.SearchResult;
import javax.naming.directory.Attributes;
import javax.naming.ldap.LdapContext;
import javax.naming.ldap.InitialLdapContext;

import org.globus.mds.gsi.common.GSIMechanism;

// we could add: aliasing, referral support
public class MyGridInfoSearch {

    //Default values
    private static final String version =
   org.globus.common.Version.getVersion();

    private static final String DEFAULT_CTX =
   "com.sun.jndi.ldap.LdapCtxFactory";

    private String hostname = "t3.itso-tupi.com";
    private int port = 2135;
    private String baseDN = "mds-vo-name=local, o=grid";
```

```
    private int scope = SearchControls.SUBTREE_SCOPE;
    private int ldapVersion = 3;
    private int sizeLimit = 0;
    private int timeLimit = 0;
    private boolean ldapTrace = false;
    private String saslMech;

    private String bindDN;
    private String password;
    private String qop = "auth"; //could be auth, auth-int, auth-conf

public MyGridInfoSearch() {
    }
```

The org.globus.mds.gsi.common.GSIMechanism() method verifies that the GSI security credentials are valid and sets the context. The search method below performs two functions: Authentication and searching. It calls the method GSIMechanism.

Example 6-9 GridInfoSearch example (2 of 4)

```
//Search the ldap server for the filter specified in the main function
private void search(String filter) {

   Hashtable env = new Hashtable();

   String url = "ldap://" + hostname + ":" + port;

   env.put("java.naming.ldap.version", String.valueOf(ldapVersion));
   env.put(Context.INITIAL_CONTEXT_FACTORY, DEFAULT_CTX);
   env.put(Context.PROVIDER_URL, url);

   if (bindDN != null) {
       env.put(Context.SECURITY_PRINCIPAL, bindDN);
   }

   //use GSI authentication from grid-proxy-init certificate
      saslMech = GSIMechanism.NAME;
      env.put("javax.security.sasl.client.pkgs",
         "org.globus.mds.gsi.jndi");

       env.put(Context.SECURITY_AUTHENTICATION, saslMech);

       env.put("javax.security.sasl.qop", qop);

   LdapContext ctx = null;
```

```
   //create a new ldap context to hold perform search on filter
   try {

       ctx = new InitialLdapContext(env, null);

       SearchControls constraints = new SearchControls();

       constraints.setSearchScope(scope);
       constraints.setCountLimit(sizeLimit);
       constraints.setTimeLimit(timeLimit);

      //store the results of the search in the results variable

       NamingEnumeration results = ctx.search(baseDN, filter, constraints);

       displayResults(results);

   } catch (Exception e) {
       System.err.println("Failed to search: " + e.getMessage());

   } finally {
       if (ctx != null) {
      try { ctx.close(); } catch (Exception e) {}
       }
   }
}
```

The above search() method uses the filter to perform an LDAP search. A hash table is used to store all of the search information, such as the version of LDAP to use, the type of security authentication to use, and the URL of the LDAP server to query. The results returned from the search are stored in a results variable, which is passed to the displayResults() method, shown in Example 6-10.

Example 6-10 GridInfoSearch example (3 of 4)

```
// DISPLAY RESULTS OF SEARCH
private void displayResults(NamingEnumeration results)
   throws NamingException {

   if (results == null) return;

   String dn;
   String attribute;
   Attributes attrs;
   Attribute at;
   SearchResult si;
```

```
   //use the results variable from search method and store them in a printable
   //variable.

   while (results.hasMoreElements()) {
       si = (SearchResult)results.next();
       attrs = si.getAttributes();

       if (si.getName().trim().length() == 0) {
      dn = baseDN;
       } else {
      dn = si.getName() + ", " + baseDN;
       }
       System.out.println("dn: " + dn);

       for (NamingEnumeration ae = attrs.getAll(); ae.hasMoreElements();) {
      at = (Attribute)ae.next();

      attribute = at.getID();

      Enumeration vals = at.getAll();
      while(vals.hasMoreElements()) {
          System.out.println(attribute + ": " + vals.nextElement());
      }
       }
       System.out.println();
   }
}
```

The displayResults() method above takes the information stored in the results variable, parses it into separate attributes, and converts it to an enumeration type so it can be printed.

Example 6-11 GridInfoSearch example (4 of 4)

```
//Create new instance of MyGridInfoSearch and use specified filter string
public static void main( String [] args ) {

   MyGridInfoSearch gridInfoSearch = new MyGridInfoSearch();
   String filter = "(&(objectclass=MdsOs)(Mds-Os-name=Linux))";
   System.out.println("Your search string is: " + filter);


   gridInfoSearch.search(filter);
    }
}
```

The above code creates a new instance of MyGridInfoSearch and passes the filter to the search method.

6.5 RSL

We introduced RSL in 2.1.2, “Resource management” on page 17. Now we show some programming examples that utilize RSL.

6.5.1 Example using RSL

Example 6-12 utilizes the org.globus.rsl package provided by the JavaCoG Kit to parse and display the RSL string.

Example 6-12 RSL example (1 of 6)

```
mport org.globus.rsl.*;

import java.util.*;
import java.io.*;

import junit.framework.*;
import junit.extensions.*;

public class MyRSL {
   public void MyRSL(){
   }

   public static void main(String[] args) {
      RslAttributes attribs;
      Map rslsubvars;

      String myrslstring =
             "&(executable=/bin/echo)(arguments=\"globusproject\")";
      String myrslstring2 =
             "&(rsl_substitution=(EXECDIR\"/bin\"))(executable=$(EXECDIR)/echo)
             (arguments=\"www.globus.org\")";
        String myrslstring3 = "globusproject";
        String myrslstring4 = "arguments";
        String myrslstring5 = "/bin/ls";

      try {
         // print attributes
         attribs = new RslAttributes(myrslstring);
         System.out.println("Your rsl string is: "+ attribs.toRSL() );
         String result = attribs.getSingle("executable");
         System.out.println("Your executable is: "+ result);
         result = attribs.getSingle("arguments");
         System.out.println("Your argument is: "+ result);
         System.out.println("");
```

The variable myrslstring contains the RSL string. The string is then stored as type RslAttributes. The RslAttributes class allows parsing, modifying, and deleting values in the string. The getSingle() method returns the value of a specified attribute.

Example 6-13 RSL example (2 of 6)

```
//remove attributes
         System.out.println("Your rsl string is: "+ attribs.toRSL() );
         attribs.remove(myrslstring4,myrslstring3);
         result = attribs.getSingle("arguments");
         System.out.println("After removing the argument "+ myrslstring3 + "
              your rsl string is: ");
         System.out.println("Your rsl string is: "+ attribs.toRSL() );
         System.out.println("");
```

Example 6-13 removes “globusproject” from the RSL string &(executable=/bin/echo)(arguments=\"globusproject\"). The remove() method finds the attribute in the string and removes the value “globusproject”. The remaining string is printed to the screen.

Example 6-14 RSL example (3 of 6)

```
//add attributes
         System.out.println("Your rsl string is: "+ attribs.toRSL() );
         attribs.add(myrslstring4,myrslstring5);
         result = attribs.getSingle("arguments");
         System.out.println("After adding the arguement "+ result + " your rsl
              string is: ");
         System.out.println( attribs.toRSL() );
         System.out.println("");
```

Example 6-14 adds a value of “www.globus.org” to the attribute argument. The add() method finds the attribute and adds the value “www.globus.org”.

Example 6-15 RSL example (4 of 6)

```
/uses rsl substitution
         attribs = new RslAttributes(myrslstring2);
         System.out.println("Your rsl string is: "+ attribs.toRSL() );
         rslsubvars = attribs.getVariables("rsl_substitution");
           if (rslsubvars.containsKey("EXECDIR")){
            rslsubvars.get("EXECDIR");
            result = attribs.getSingle("executable");
            System.out.println("Your executable is: "+ result);
            System.out.println("");
           }
```

Example 6-15 on page 146 uses rsl_substitution to create variables within the RSL string. The getVariables() method gets all of the variables declared within rsl_substitution, while the get() method gets the value for the specified variable. In this case the value for the variable EXECDIR is “/bin”.

Example 6-16 RSL example (5 of 6)

```
//add new rsl string
         ListRslNode rslTree = new ListRslNode(RslNode.AND);
         NameOpValue nv = null;
         List vals = null;

         rslTree.add(new NameOpValue("executable",
                NameOpValue.EQ,
                "/bin/date"));

         rslTree.add(new NameOpValue("maxMemory",
                NameOpValue.LTEQ,
                "5"));

         rslTree.add(new NameOpValue("arguments",
                NameOpValue.EQ,
                new String [] {"+%H", "+%M", "%S "}));

         nv = rslTree.getParam("EXECUTABLE");
         System.out.println("The executable you have added is: "+ nv);

         nv = rslTree.getParam("MAXMEMORY");
         System.out.println("The memory you have added is: "+ nv);

         nv = rslTree.getParam("ARGUMENTS");
         System.out.println("The arguments you have added is: "+ nv);
         System.out.println("");
```

Example 6-16 uses the ListRslNode class to create attributes. The add() method creates a new RSL string. In this case the RSL string contains the executable, /bin/date; the maxMemory, 5 MB; and arguments, +%H +%M +%S. These values are then stored in NameOpValue.

Example 6-17 RSL example (6 of 6)

```
//remove attribute from string
         ListRslNode node = null;
         attribs = new RslAttributes(myrslstring2);
         System.out.println("Your rsl string is: "+ attribs.toRSL() );

         try {
           node = (ListRslNode)RSLParser.parse(ListRslNode.class,
                 myrslstring2);
```

```
        } catch(Exception e) {
              System.out.println("Cannot parse rsl string");
        }
          nv = node.removeParam("arguments");
                        vals = nv.getValues();
                 System.out.println("Removing " + nv);
                System.out.println("Your string with the arguemnts removed: " +
                    node);

      catch (Exception e) {
         System.out.println("Cannot parse rsl string");
      }
   }
}
```

Example 6-17 on page 147 stores an RSL string as a ListRslNode and removes the argument attribute from the string. The removeParam() method removes the arguments attribute and all of its variables.

6.6 GridFTP

The JavaCoG Kit provides the org.globus.ftp package to perform FTP and GridFTP operations. It is basically an implementation of the FTP and GridFTP protocol.

The FTP client provides the following functionality:

- Client/server FTP file transfer
- Third-party file transfer
- Passive and active operation modes
- ASCII/IMAGE data types
- Stream transmission mode

The GridFTP client extends the FTP client by providing the following additional capabilities:

- Extended block mode
- Parallel transfers
- Striped transfers
- Restart markers
- Performance markers

Packages

The following packages are available to be used with the Java CoG:

- org.globus.ftp (classes for direct use)

- org.globus.ftp.vanilla (Vanilla FTP protocol)
- org.globus.ftp.extended (GridFTP protocol)
- org.globus.ftp.dc (data channel functionality)
- org.globus.ftp.exception (exceptions)

6.6.1 GridFTP basic third-party transfer

Example 6-18 demonstrates how to perform a third-party file transfer using extended block mode and GSI security using the GridFTP protocol.

In order to transfer a file from one server to another we need to create a client on each server. In order to change any FTP client settings like Mode or Security, the issuer must authenticate to the FTP client using its credentials.

Example 6-18 GridFTP basic third-party transfer (1 of 4)

```
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.globus.ftp.DataChannelAuthentication;
import org.globus.ftp.GridFTPClient;
import org.globus.ftp.GridFTPSession;
import org.globus.gsi.GlobusCredential;
import org.globus.gsi.gssapi.GlobusGSSCredentialImpl;
import org.ietf.jgss.GSSCredential;

/**
 * GridFTPthird
 *
 * Performs a server to server GridFTP operation
 */
public class GridFTPthird {

   private GridFTPClient sClient = null;//source FTPClient
   private GridFTPClient dClient = null;//destination FTPClient
   private GSSCredential cred = null;
```

In Example 6-19 we will read the credentials from the proxy file. For authentication we need a GSSCredential object, so we have to change the GlobusCredential object to GSSCredential. By doing that it is important to use the DEFAULT_LIFETIME flag.

Example 6-19 GridFTP basic third-party transfer (2 of 4)

```
   //Load credentials from proxy file
   private void getCredentials() throws Exception {

      GlobusCredential gcred = new GlobusCredential("/tmp/x509up_u1101");
      System.out.println("GCRED: "+gcred.toString());
```

```
      cred = new GlobusGSSCredentialImpl(gcred,
GSSCredential.DEFAULT_LIFETIME);
   }
```

When creating the GridFTPClient it is important to use the GridFTP port, which defaults to 2811. Authentication is done using the authenticate() method, passing the GSSCredentials. It is important to authenticate to the GridFTPClient first, before setting or changing any other properties like transfer-type or mode.

Setting the client manually to active or passive is possible, but not required for third-party transfers.

Example 6-20 GridFTP basic third-party transfer (3 of 4)

```
//Initializing the FTPClient on the source server
private void initSourceClient() throws Exception {

   sClient = new GridFTPClient("t1.itso-tupi.com", 2811);

   sClient.authenticate(cred);//authenticating
   sClient.setProtectionBufferSize(16384);//buffersize
   sClient.setType(GridFTPSession.TYPE_IMAGE);//transfertype
   sClient.setMode(GridFTPSession.MODE_EBLOCK);//transfermode
   sClient.setDataChannelAuthentication(DataChannelAuthentication.SELF);
   sClient.setDataChannelProtection(GridFTPSession.PROTECTION_SAFE);
}
//Initializing the FTPClient on the destination server
private void initDestClient() throws Exception {

   dClient = new GridFTPClient("t2.itso-tupi.com", 2811);
   dClient.authenticate(cred);
   dClient.setProtectionBufferSize(16384);
   dClient.setType(GridFTPSession.TYPE_IMAGE);
   dClient.setMode(GridFTPSession.MODE_EBLOCK);
   dClient.setDataChannelAuthentication(DataChannelAuthentication.SELF);
   dClient.setDataChannelProtection(GridFTPSession.PROTECTION_SAFE);

}
```

Finally we will start the transfer, defining the source and target files.

Example 6-21 GridFTP basic third-party transfer (4 of 4)

```
   private void start() throws Exception {
      System.out.println("Starting Transfer");
      sClient.transfer(
         "/etc/hosts",
         dClient,
```

```
            "/tmp/ftpcopy.test",
            false,
            null);

         System.out.println("Finished Transfer");
      }
      public static void main(String[] args) {
         GridFTPthird ftp = new GridFTPthird();
         try {
            ftp.getCredentials();
            ftp.initDestClient();
            ftp.initSourceClient();
            ftp.start();
         } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
         }
      }
}
```

6.6.2 GridFTP client-server

When transferring a file from a local client to a server or from a server to the client, a local interface to the file storage must be supplied. The toolkit provides two interfaces: DataSink for receiving a file, and ataSource for sending a file.

Example 6-22 GridFTP client-server example (1 of 6)

```
import org.globus.ftp.DataChannelAuthentication;
import org.globus.ftp.GridFTPClient;
import org.globus.ftp.GridFTPSession;
import org.globus.gsi.GlobusCredential;
import org.globus.gsi.gssapi.GlobusGSSCredentialImpl;
import org.ietf.jgss.GSSCredential;
import org.globus.ftp.*;
import java.io.*;

/**
 * GridFTPclient
 *
 * Treansfers a file from the client to the server
 */
public class GridFTPclient {

   private GridFTPClient client = null; //Grid FTP Client
   private GSSCredential cred = null; //Credentials
```

First we have to get the credentials from our proxy file.

Example 6-23 GridFTP client-server example (2 of 6)

```
//Load credentials from proxy file
public void getCredentials() throws Exception {

   GlobusCredential gcred = new GlobusCredential("/tmp/x509up_u1101");
   System.out.println("GCRED: " + gcred.toString());
   cred =
      new GlobusGSSCredentialImpl(gcred, GSSCredential.DEFAULT_LIFETIME);
}
```

We create a GridFTPClient on the remote host, authenticate, and set the parameters.

Example 6-24 GridFTP client-server example (3 of 6)

```
//Initializes the ftp client on given host
public void createFTPClient(String ftphost) throws Exception {
   client = new GridFTPClient(ftphost, 2811);
   client.authenticate(cred); //authenticating
   client.setProtectionBufferSize(16384); //buffersize
   client.setType(GridFTPSession.TYPE_IMAGE); //transfertype
   client.setMode(GridFTPSession.MODE_EBLOCK); //transfermode
   client.setDataChannelAuthentication(DataChannelAuthentication.SELF);
   client.setDataChannelProtection(GridFTPSession.PROTECTION_SAFE);
}
```

To send a file to a server we have to provide an interface to our local file. This can be done using the DataSource interface, as shown in Example 6-25, or using the DataSourceStream. Note, however, that DataSourceStream does not work with extended block mode. As we are using the extended block mode, we have to use the DataSource interface.

Example 6-25 GridFTP client-server example (4 of 6)

```
   public void ClientToServer(String localFileName, String remoteFileName)
      throws Exception {

      DataSource datasource = null;
      datasource = new FileRandomIO(new
java.io.RandomAccessFile(localFileName, "rw"));

      client.extendedPut(remoteFileName, datasource, null);
   }
```

When receiving a file from a server we have to provide a local file interface to write the data to. In this case it is the DataSink interface. Again, if extended block mode is not used, the DataSinkStream can be used instead.

Attention: By default the GridFTPClient is passive, so it can receive files. As we are going to use the same GridFTPClient to send data, we have to set it to active, and our local to passive. This can be done using:

```
client.setLocalPassive();
client.setActive();
```

Note that the passive side must always be set first.

Example 6-26 GridFTP client-server example (5 of 6)

```
   public void ServerToClient(String localFileName, String remoteFileName)
      throws Exception {
      long size = client.getSize(remoteFileName);
      DataSink sink = null;
      sink = new FileRandomIO(new java.io.RandomAccessFile(localFileName,
"rw"));

      //setting FTPClient to active so be able to send file
      client.setLocalPassive();
      client.setActive();

      client.extendedGet(remoteFileName, size, sink, null);
   }
```

If performance or progress monitoring is required, it can be easily implemented using the MarkerListener interface. See the JavaCoG API description for more information.

Example 6-27 GridFTP client-server example (6 of 6)

```
   public static void main(String[] args) {

      try{
      //Initialize
      GridFTPclient gftp = new GridFTPclient();

      //get credentials
      System.out.println("Getting Credentials");
      gftp.getCredentials();

      //get ftp client
      System.out.println("Creating the FTP Client");
      gftp.createFTPClient("d1.itso-dakota.com");

      //perform client to server copy
      System.out.println("Tranfering Client to Server");
      gftp.ClientToServer("/tmp/d2-to-d1", "/tmp/d2-to-d1");
```

```
      //perform server to client copy
      System.out.println("Transfering Server to Client");
      gftp.ServerToClient("/tmp/d1-to-d2", "/tmp/d1-to-d2");

      System.out.println("All Done");
      }catch(Exception e){
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

6.6.3 URLCopy

The URLCopy class provides a very easy way of transferring files. It understands the GSIFTP, GASS, FTP, and FILE protocol.

Example 6-28 URLCopy example (1 of 2)

```
package test;

import org.globus.io.urlcopy.UrlCopy;
import org.globus.io.urlcopy.UrlCopyListener;
import org.globus.util.GlobusURL;
import org.globus.gsi.gssapi.auth.*;

/**
 * URLCopy
 *
 * Performs a copy based on the URLCopy package
 */
public class URLCopy implements UrlCopyListener{

   public void transfer(int transferedBytes, int totalBytes){
      System.out.println("Transferred "+transferedBytes+" of "+totalBytes + "
Bytes");
   }
   public void transferCompleted(){
      System.out.println("Transfer Complete");
   }
   public void transferError(Exception e){
      System.out.println("Error: "+e.getMessage());
   }
```

All we need to do is to assign the URLCopy object properties like DestinationUrl and SourceAuthorization. If the transfer is a third-party transfer, then the flag must be set using ucopy.setUseThirdPartyCopy(true).

Example 6-29 URLCopy example (2 of 2)

```
   public void ucopy(){
      try{
      UrlCopy ucopy = new UrlCopy();
      GlobusURL durl = new
GlobusURL("gsiftp://t2.itso-tupi.com//tmp/urlcopy");
      GlobusURL surl = new GlobusURL("gsiftp://t1.itso-tupi.com//etc/hosts");
      Authorization srcAuth = null;
      Authorization dstAuth = null;

      dstAuth = new
IdentityAuthorization("/O=Grid/O=Globus/CN=host/t2.itso-tupi.com");
      srcAuth  = new
IdentityAuthorization("/O=Grid/O=Globus/CN=host/t1.itso-tupi.com");

      //ucopy.addUrlCopyListener(this);
      ucopy.setDestinationUrl(durl);
      ucopy.setSourceUrl(surl);
      ucopy.setUseThirdPartyCopy(true);
      ucopy.setSourceAuthorization(srcAuth);
      ucopy.setDestinationAuthorization(dstAuth);
      System.out.println("Start Copy()");
      ucopy.copy();

      System.out.println("All done");
      }catch(Exception e){
         System.out.println("Error: "+e.getMessage());
      }
   }
   public static void main(String[] args) {

      URLCopy u = new URLCopy();
      u.ucopy();
   }
}
```

6.7 GASS

The GASS API can be used to send or retrieve data, files, or application output. When, for example, submitting a job in batch mode, the result of the job can be picked up using the GASS API or any standard binary tool provided by the Globus Toolkit. When using interactive job submission, the GASS API can be used to retrieve the output of an application by redirecting standard out and standard error to the client.

These two examples will show how to submit a job and retrieve the results:

- GASS Batch
- GASS Interactive

6.7.1 Batch GASS example

The following examples are of batch GASS.

Example 6-30 Batch GASS example (1 of 4)

```
import org.globus.gram.Gram;
import org.globus.gram.GramJob;
import org.globus.gram.GramJobListener;
import org.globus.io.gass.server.GassServer;
import org.globus.util.deactivator.Deactivator;

/**
 * Example of using GRAM & GASS in batch mode
 *
 */
public class GASSBatch implements GramJobListener{

   private GassServer gServer = null;
   private String gURL = null;
   private String JobID = null;

   //To Start the GASS Server
   private void startGassServer() {

      try {
         gServer = new GassServer(0);
         gURL = gServer.getURL();
      } catch (Exception e) {
         System.out.println("GassServer Error" + e.getMessage());
      }
      gServer.registerDefaultDeactivator();

      System.out.println("GassServer started...");
   }
```

Starting the GASS server and getting the server URL will provide us with the ability to retrieve data later. By registering the default deactivator we can destroy the GASS server before we exit the program.

Example 6-31 Batch GASS example (2 of 4)

```
//Method called by the GRAMJobListener
   public void statusChanged(GramJob job) {
```

```
      System.out.println(
         "Job: "
            + job.getIDAsString()
            + " Status: "
            + job.getStatusAsString());
   }
   private synchronized void runJob() {
      //RSL String to be executed
      String RSL = "&(executable=/bin/ls)(directory=/bin)(arguments=-l)";
      String gRSL = null;
      //Resource Manager Contact
      String rmc = "t2.itso-tupi.com";

      gRSL =
         RSL
            + "(stdout=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stdout test)"
            + "(stderr=x-gass-cache://$(GLOBUS_GRAM_JOB_CONTACT)stderr test)";

      //Instantiating the GramJob with the RSL
      GramJob job = new GramJob(gRSL);
      job.addListener(this);
```

Our RSL string that will execute an application needs to be modified, so the standard out and error is written to the GASS server.

Example 6-32 Batch GASS example (3 of 4)

```
      System.out.println("Requesting Job...");
      try {
         job.request(rmc);
      } catch (Exception e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
```

We request the job and deactivate GASS using the deactivator when the job is done.

Example 6-33 Batch GASS example (4 of 4)

```
   public static void main(String[] args) {

      System.out.println("Starting GRAM & GASS in Batch mode...");

      GASSBatch run = new GASSBatch();
      run.startGassServer();
```

```
        run.runJob();
        System.out.println("Job Sumbitted Done.");
        Deactivator.deactivateAll();
    }
}
```

6.7.2 Interactive GASS example

In interactive mode we will reroute the output of the application to our client, instead of storing it.

Example 6-34 Interactive GASS example (1 of 3)

```
import org.globus.gram.Gram;
import org.globus.gram.GramJob;
import org.globus.gram.GramJobListener;
import org.globus.io.gass.server.GassServer;
import org.globus.io.gass.server.JobOutputListener;
import org.globus.io.gass.server.JobOutputStream;
import org.globus.util.deactivator.Deactivator;

/**
 * Example of using GRAM & GASS in interactive mode
 *
 */
public class GASSInteractive implements GramJobListener, JobOutputListener {

    private GassServer gServer = null;
    private String gURL = null;
    private JobOutputStream oStream = null; //OutputStream
    private JobOutputStream eStream = null; //ErrorStream
    private String JobID = null;

    //To Start the GASS Server
    private void startGassServer() {

        try {
            gServer = new GassServer(0);
            gURL = gServer.getURL();
        } catch (Exception e) {
            System.out.println("GassServer Error" + e.getMessage());
        }
        gServer.registerDefaultDeactivator();

        // job output vars
        oStream = new JobOutputStream(this);
        eStream = new JobOutputStream(this);
        JobID = String.valueOf(System.currentTimeMillis());
```

```
      // register output listeners
      gServer.registerJobOutputStream("out-" + JobID, oStream);
      gServer.registerJobOutputStream("err-" + JobID, eStream);
      System.out.println("GassServer started...");
   }
```

We register listeners to the GASS server so that we can receive the output of the application, and also know when the application is finished.

The method outputChanged() will provide the screen output of the application line by line. In order to display it we can just reroute it to our screen.

The method outputClosed() tells us that there will be no more output from the application.

Example 6-35 Interactive GASS example (2 of 3)

```
   //Method called by the JobOutputListener
   public void outputChanged(String output) {
      System.out.println("JobOutput: " + output);
   }
   //Method called by the JobOutputListener
   public void outputClosed() {
      System.out.println("JobOutput: OutputClosed");
   }
   //Method called by the GRAMJobListener
   public void statusChanged(GramJob job) {
      System.out.println(
         "Job: "
            + job.getIDAsString()
            + " Status: "
            + job.getStatusAsString());

      if (job.getStatus() == GramJob.STATUS_DONE) {
         synchronized (this) {
            notify();
         }
      }
   }
```

Again we enhance the RSL string so that the output will be rerouted to our client, and finally we run the application.

Example 6-36 Interactive GASS example (3 of 3)

```
   private synchronized void runJob() {
```

```
    //RSL String to be executed
    String RSL = "&(executable=/bin/ls)(directory=/bin)(arguments=-l)";
    String gRSL = null;

    //Resource Manager Contact
    String rmc = "t2.itso-tupi.com";

    gRSL =
       "&"
          + RSL.substring(0, RSL.indexOf('&'))
          + "(rsl_substitution=(GLOBUSRUN_GASS_URL "
          + gURL
          + "))"
          + RSL.substring(RSL.indexOf('&') + 1, RSL.length())
          + "(stdout=$(GLOBUSRUN_GASS_URL)/dev/stdout-"
          + JobID
          + ")"
          + "(stderr=$(GLOBUSRUN_GASS_URL)/dev/stderr-"
          + JobID
          + ")";

    //Instantiating the GramJob with the RSL
    GramJob job = new GramJob(gRSL);
    job.addListener(this);

    try {
       Gram.ping(rmc);
    } catch (Exception e) {
       System.out.println("Ping Failed: " + e.getMessage());
    }

    System.out.println("Requesting Job...");
    try {
       job.request(rmc);

    } catch (Exception e) {
       System.out.println("Error: " + e.getMessage());
    }
    //wait for job completion
    synchronized (this) {
       try {
          wait();
       } catch (InterruptedException e) {
          System.out.println("Error: " +e.getMessage());
       }
    }
}
public static void main(String[] args) {
    System.out.println("Starting GRAM & GASS interactive...");
```

```
        GASSInteractive run = new GASSInteractive();
        run.startGassServer();
        run.runJob();
        System.out.println("All Done.");
        Deactivator.deactivateAll();
    }
}
```

6.8 Summary

In this chapter we have provided several programming examples of using the Java CoG to access the various services provided by the Globus Toolkit.

By using these examples and the sample code provided with the Java CoG, readers can gain an understanding of the various Java classes provided by the CoG and start utilizing them to create their own applications.

7

Using Globus Toolkit for data management

There are two major components for data management in the Globus Toolkit:

- Data transfer and access
- Data replication and management

For basic data transfer and access, the toolkit provides the Globus Access to Secondary Storage (GASS) module, which allows applications to access remote data by specifying a URL.

For high-performance and third-party data transfer and access, Globus Toolkit Version 2 implements the GridFTP protocol. This protocol is based on the IETF FTP protocol, and adds extensions for partial file transfer, striped/parallel file segment transfer, TCP buffer control, progress monitoring, and extended restart.

© Copyright IBM Corp. 2003. All rights reserved.

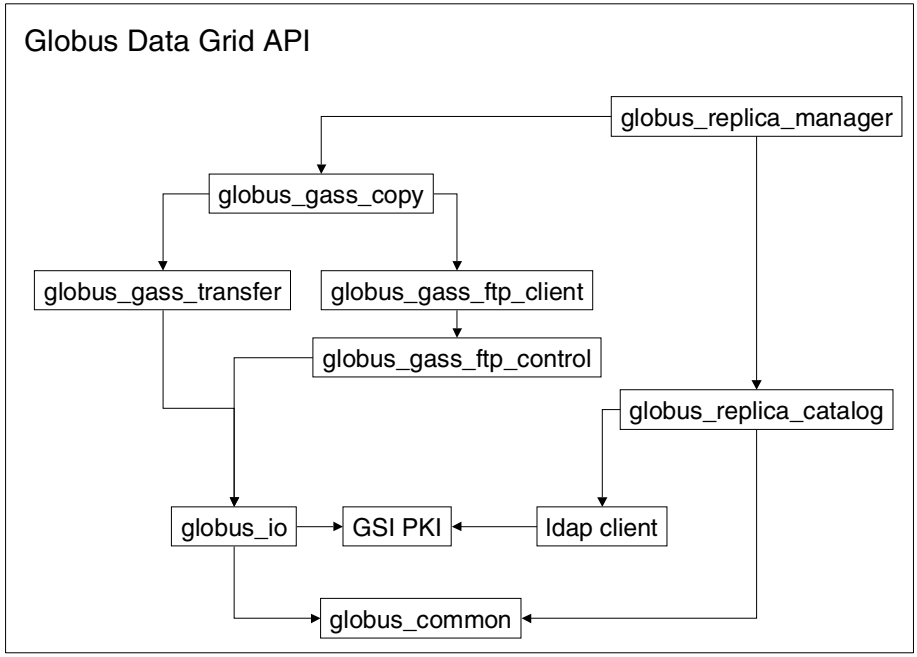


Figure 7-1 Data management interfaces

Figure 7-1 provides a view of the various modules associated with data management in the Globus Toolkit and how they relate to one another. These modules are described in more detail throughout this chapter.

7.1 Using a Globus Toolkit data grid with RSL

The Global Access to Secondary Storage is a simple, multi-protocol transfer software integrated with GRAM. The purpose of GASS is to provide a simple way to enable grid applications to securely stage and access data to and from remote file servers using a simple protocol-independent API. GASS features can easily be used via the RSL language describing the job submission.

By using URLs to specify file names, RSL permits jobs to work on remotely stored files. GASS transparently manages the data movement. Using https:// or http:// prefixes in a URL connects to a remote GASS server, and using gsiftp:// as a prefix, connects to gsiftp servers.

All files specified as input parameters are copied to each node, so each node works on its local copy. If multiple jobs output data to the same file, the data is appended to the file.

Table 7-1 is a list of RSL attributes that are used to stage files in and out.

Table 7-1 Data movement RSL-specific attributes

Attributes	Description
executable	The name of the executable file to run on the remote machine. If the value is a GASS URL, the file is transferred to the remote GASS cache before executing the job and removed after the job has terminated.
file_clean_up	Specifies a list of files that will be removed after the job is completed.
file_stage_in	Specifies a list of ("remote URL" "local file") pairs that indicate files to be staged to the nodes that will run the job.
file_stage_in_shared	Specifies a list of ("remote URL" "local file") pairs that indicate files to be staged into the cache. A symbolic link from the cache to the "local file" path will be made.
file_stage_out	Specifies a list of ("local file" "remote URL") pairs that indicate files to be staged from the job to a GASS-compatible file server.
gass_cache	Specifies location to override the GASS cache location (~/.globus/.gass-cache by default).

Attributes	Description
remote_io_url	Writes the given value (a URL base string) to a file, and adds the path to that file to the environment through the GLOBUS_REMOTE_IO_URL environment variable. If this is specified as part of a job restart RSL, the job manager will update the file's contents. This is intended for jobs that want to access files via GASS, but the URL of the GASS server has changed due to a GASS server restart.
stdin	The name of the file to be used as standard input for the executable on the remote machine. If the value is a GASS URL, the file is transferred to the remote GASS cache before executing the job, and removed after the job has terminated.
stdout	The name of the remote file to store the standard output from the job. If the value is a GASS URL, the standard output from the job is transferred dynamically during the execution of the job.
stderr	The name of the remote file to store the standard error from the job. If the value is a GASS URL, the standard error from the job is transferred dynamically during the execution of the job.

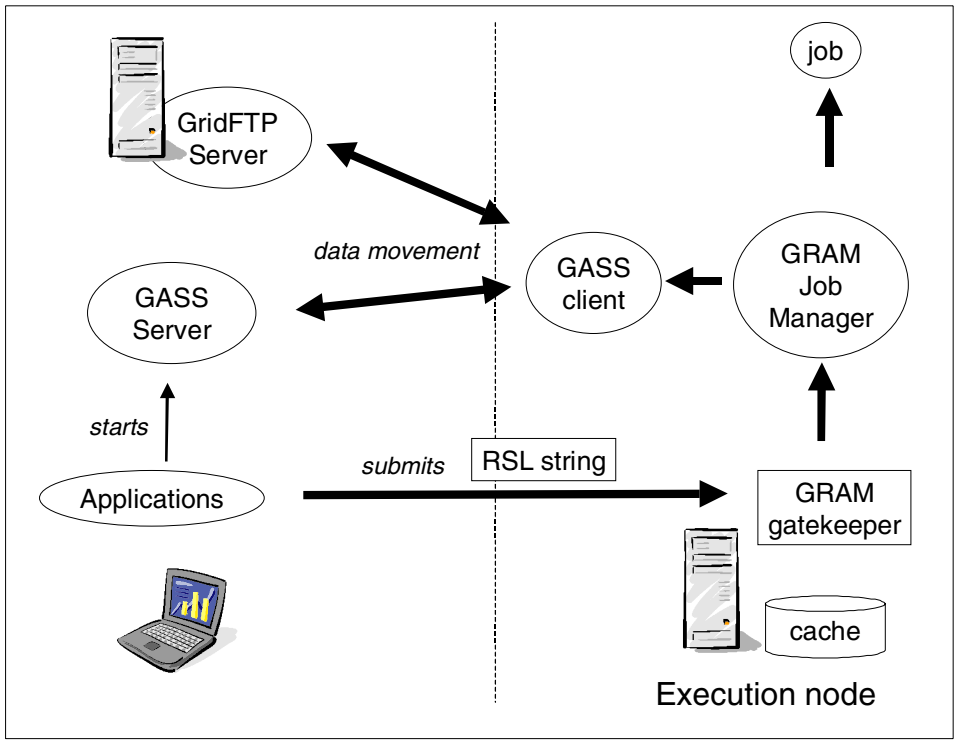


Figure 7-2 File staging

Let us consider an example where a program called MyProg generates an output file named OutputFileGenerated on the execution node. This output file is retrieved from the execution node and saved as /tmp/RetrievedFile on the machine where the **globusrun** command was issued.

Example 7-1 Staging files out with RSL

```
globusrun -o -s -r t2 '&(executable=/bin/MyProg) (arguments=-l) (count=1)
(file_stage_out=(OutputFileGenerated $(GLOBUSRUN_GASS_URL)/tmp/RetrievedFile))'
```

$(GLOBUSRUN_GASS_URL) is automatically expanded to the URL of the local GASS server started when **globusrun** is issued. This GASS server is started locally by using the -s option, and is used when access to files stored on the submission node is requested.

Example 7-2 on page 168 is a similar example, but the output file is put on a GridFTP server running on b1.

Example 7-2 Using GridFTP in RSL

```
globusrun -o -r t2 '&(executable=/bin/MyProg) (arguments=-l) (count=1)
(file_stage_out=(OutputFileGenerated gsiftp://b1/tmp/RetrievedFileOnb1))'
```

The file_stage_in directive performs the opposite task. It can move data from one location to the execution node. In the following examples, the FileCopiedOnTheExecutionNodes is copied into the home directory of the user used for the job execution on the execution node and used by the Exec program. The second example uses the local GASS server started by **globusrun**.

Example 7-3 Staging files in

```
globusrun -o -s -r a1 '&(executable=Exec) (arguments=-l) (count=1)
(file_stage_in=(gsiftp://m0/tmp/files_on_storage_server
FileCopiedOnTheExecutionNodes))'

globusrun -o -s -r t1 '&(executable=Exec) (arguments=-l) (count=1)
(file_stage_in=($(GLOBUSRUN_GASS_URL)/local_file_on_the_submission_node
FileCopiedOnTheExecutionNodes))'
```

You can use file_stage_in_shared to copy the file in the GASS cache directory. Only a symbolic link to the file will be created.

Example 7-4 Example using file_stage_in_shared

```
[globus@m0 globus]$  globusrun -o -s -r a1 '&(executable=/bin/ls)
(arguments=-l) (count=1) (file_stage_in_shared=(gsiftp://m0/tmp/File NewFile))
(count=1)'
total 748
lrwxrwxrwx    1 muser    mgroup        122 Mar 14 00:03 NewFile->
/home/muser/.globus/.gass_cache/local/md5/3b/66/18/bd493014532754516a612f2ac6/m
d5/cc/3f/1d/aae03be0ceb81e214e2a449ac3/data
```

If a file is already there, the job will fail.

Example 7-5 Example of failure

```
[globus@m0 globus]$  globusrun -o -s -r a1 '&(executable=/bin/ls)
(arguments=-l) (file_stage_in=(gsiftp://m0/tmp/0 NewFile)) (count=1)'
GRAM Job failed because the job manager could not stage in a file (error code
135)
```

You can use file_clean_up to fix this problem and delete all files that were staged during the job execution.

7.2 Globus Toolkit data grid low-level API: globus_io

To use this API, you must activate the GLOBUS_IO module in your program:

```
globus_module_activate(GLOBUS_IO_MODULE)
```

Note: The complete Globus IO API documentation is available from the Globus project Web site at the following URL:

http://www-unix.globus.org/api/c-globus-2.2/globus_io/html/index.html

The globus_io library was motivated by the desire to provide a uniform I/O interface to stream and datagram style communications. It provides wrappers for using TCP and UDP sockets and file I/O.

The Globus Toolkit 2.2 uses a specific handle to refer to a file. It is defined as globus_io_handle_t.

Two functions are provided to retrieve I/O handles:

- globus_io_file_posix_convert(), which can convert a normal file descriptor into a Globus Toolkit 2.2 handle
- globus_io_file_open(), which creates a Globus Toolkit 2.2 handle from a file name

The Globus Toolkit 2.2 provides I/O functions that map the POSIX systems calls and use the Globus Toolkit 2.2 file handle as a parameter instead of the file descriptor.

- globus_io_read(), globus_io_write()
- globus_io_writev() for vectorized write operations

Globus Toolkit 2.2 provides an asynchronous or non-blocking I/O that uses a callback mechanism. The callback is a function given as a parameter to the globus_io calls that will be called when the operation has completed. By using condition variables, the call back can alert the process that the operation has completed.

- globus_io_register_read(),globus_io_register_write()
- globus_io_register_writev()

The Globus Toolkit 2.2 provides functions to manipulate socket attributes, and by doing so extends the POSIX system sockets calls. In particular, it provides a set of functions, globus_io_attr_*(), that are used to establish authentication and authorization at the socket level (see Example 7-12 on page 173 and Example 7-13 on page 176).

An Internet socket is described as a globus_io_handle_t structure in the globus_io API. This handle is created when calling the followings Globus functions:

- globus_io_tcp_create_listener()
- globus_io_tcp_accept(), globus_io_tcp_register_accept()
- globus_io_tcp_connect()

These functions are respectively equivalent to the listen(), accept(), and connect() POSIX system calls. globus_io_tcp_register_accept() is the asynchronous version of globus_io_tcp_accept().

The Globus API adds authorization, authentication, and encryption features to the normal POSIX sockets via GSI and OpenSSL libraries. A handle of type globus_io_secure_authorization_data_t is used to manipulate these additional security attributes. It needs to be initialized via globus_io_secure_authorization_data_initialize() before being used in other functions.

- globus_io_attr_set_secure_authentication_mode() is used to determine whether to call the GSSAPI security context establishment functions once a socket connection is established. A credential handle is provided to the function and needs to be initialized before it is used. See the getCredential() function in Example 7-12 on page 173.

Example 7-6 Activating GSSAPI security on a socket communication

```
globus_io_attr_set_secure_authentication_mode(
      &io_attr,       //globus_io_handle_t
      GLOBUS_IO_SECURE_AUTHENTICATION_MODE_GSSAPI,  // use GSI
      credential_handle));
```

- globus_io_attr_set_secure_authorization_mode() is used to determine what security identities to authorize as the peer-to-security handshake that is done when making an authenticated connection. The functions take both a globus_io handle and a Globus secure attribute handle.

  The mode is specified in the second argument. GLOBUS_IO_SECURE_AUTHORIZATION_MODE_SELF authorizes any connection with the same credentials as the local credentials used when creating this handle.

For the complete list of available authorization modes, see

http://www-unix.globus.org/api/c-globus-2.2/globus_io/html/group__security.html#a7

Example 7-7 globus_io_attr_set_secure_authorization_mode()

```
globus_io_attr_set_secure_authorization_mode(
```

```
        &io_attr,     //globus_io_handle_t
        GLOBUS_IO_SECURE_AUTHORIZATION_MODE_SELF,
        &auth_data)
```

- globus_io_attr_set_secure_channel_mode() is used to determine if any data wrapping should be done on the socket connection. GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP indicates that data protection is provided, with support for GSI features, such as delegation.

Example 7-8 globus_io_attr_set_secure_channel_mode()

```
globus_io_attr_set_secure_channel_mode(
      &io_attr,    //globus_io_handle_t
      GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP);
```

- globus_io_attr_set_secure_protection_mode() is used to determine if any data protection should be done on the socket connection. Use GLOBUS_IO_SECURE_PROTECTION_MODE_PRIVATE for encrypted messages, GLOBUS_IO_SECURE_PROTECTION_MODE_SAFE to only check the message integrity, and GLOBUS_IO_SECURE_PROTECTION_MODE_NONE for no protection.

Example 7-9 Encrypted sockets

```
globus_io_attr_set_secure_protection_mode(
      &io_attr,    //globus_io_handle_t
      GLOBUS_IO_SECURE_PROTECTION_MODE_PRIVATE);
```

- globus_io_attr_set_secure_delegation_mode() is used to determine whether the process' credentials should be delegated to the other side of the connection. GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY delegates full credentials to the server.

Example 7-10 Delegation mode

```
globus_io_attr_set_secure_delegation_mode(
      &io_attr,    //globus_io_handle_t
      GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY);
```

- globus_io_attr_set_secure_proxy_mode() is used to determine whether the process should accept limited proxy certificates for authentication. Use GLOBUS_IO_SECURE_PROXY_MODE_MANY to accept any proxy as a valid authentication.

Example 7-11 globus_io_set_secure_proxy_mode()

```
globus_io_attr_set_secure_proxy_mode(
      &io_attr,    //globus_io_handle_t
```

```
GLOBUS_IO_SECURE_PROXY_MODE_MANY);
```

7.2.1 globus_io example

In this example we establish a secure and authenticated communication between two hosts by using the globus_io functions. We submit from host m0.itso-maya.com a job (gsiclient2) to t2.itso-tupi.com that will try to communicate with a server already running on m0.itso-maya.com (gsiserver2). This process will print `Hello World` as soon as the message is received. The two processes will use mutual authentication, which means that they need to run with the same credentials on both hosts. By using the gatekeeper, the submitted job will use the same credentials as the user that submitted the job. The communication will be securely authenticated between the two hosts. The communication is also encrypted: We use the globus_io_attr_set_secure_protection_mode() call to activate encryption.

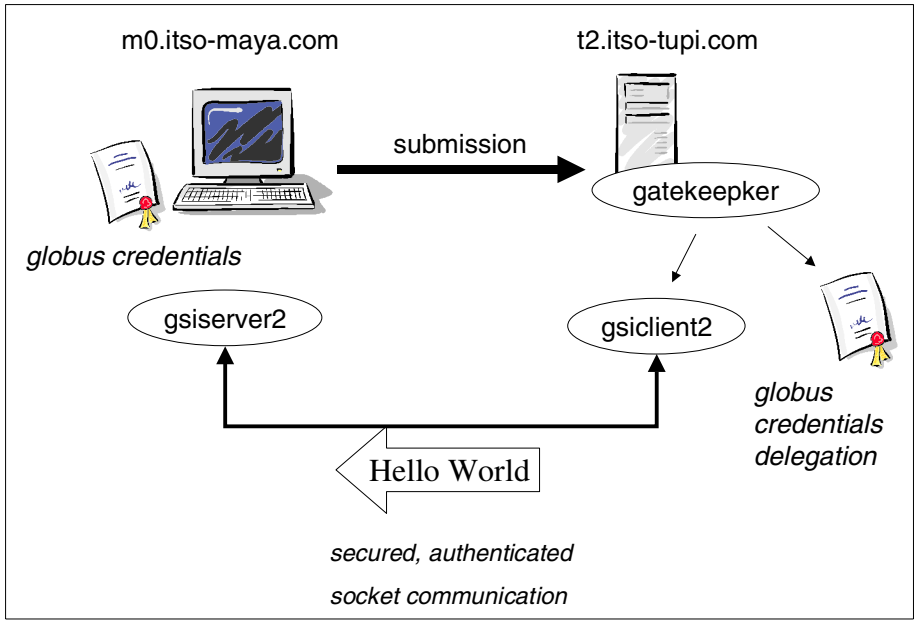


Figure 7-3 Using globus_io for secure communication

To compile the two programs:

1. First create the Makefile header with:

```
globus-makefile-header -flavor gcc32 globus_io globus_gss_assist >
globus_header
```

2. Use the following Makefile to compile:

```
include globus_header

all: gsi gsisocketclient gsisocketserver

gsisocketclient: gsisocketclient.C
   g++ -g -o gsisocketclient $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS)
gsisocketclient.C $(GLOBUS_PKG_LIBS)

gsisocketserver: gsisocketserver.C
   g++ -g -o gsisocketserver $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS)
gsisocketserver.C $(GLOBUS_PKG_LIBS)
```

3. Start the monitoring program on m0.itso-maya.com by issuing:

```
./gsisocketserver
```

4. Submit the job on t2.itso-tupi.com:

```
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Mon Mar  3 23:37:23 2003
[globus@m0 globus]$ gsiscp gsiclient2 t2.itso-tupi.com:.
gsiclient2           100% |*************************************|   126
KB    00:00
[globus@m0 globus]$ globusrun -o -r t2
'&(executable=/home/globus/gsisocketclient)
(arguments=http://m0.itso-maya.com:10000)'
m0.itso-maya.com
10000
23
```

   On the monitoring side you should see:

```
[globus@m0 globus]$ ./gsisocketserver
Hello world (secured) !@23
```

7.2.2 Skeleton source code for creating a simple GSI socket

Below we review the skeleton source code for creating a simple GSI socket.

Example 7-12 globus-io client - gsisocketclient.C

```
#include <iostream>
#include <globus_io.h>
#include "globus_gss_assist.h"
#include <string>

// macro to use a C++ string class as a char* parmeter in a C function
```

```
#define STR(a) const_cast<char*>(a.c_str())

//****************************************************************************
// This macro is defined for debugging reasons.  It checks the status of the
// globus calls and displays the Globus error message
// The t variable needs to be defined before
// ***************************************************************************
#define _(a) t=a;\
   if (t!=GLOBUS_SUCCESS) {\
                cerr <<
globus_object_printable_to_string(globus_error_get(t));\
       }
// ***************************************************************************


// ***************************************************************************
// This function is used to check the validity of the local credentials
// probably generated by gatekeeper or gsi ssh server
// ***************************************************************************
bool getCredential(gss_cred_id_t* credential_handle) {
   OM_uint32 major_status;
   OM_uint32 minor_status;

   major_status = globus_gss_assist_acquire_cred(&minor_status,
                               GSS_C_INITIATE, /* or GSS_C_ACCEPT */
                               credential_handle);

        if (major_status != GSS_S_COMPLETE)
      return false;
   else
      return true;
}

// ***************************************************************************
// ***************************************************************************
// Here is the main program: create a socket, connect to the server and say
Hello
// _() macro is used to check the error code of each Globus function and
//      display the Globus Error message
// The first argument will be used to indicate the server to connect
// to, for example http://m0.itso-maya.com:10000
//
*****************************************************************************
main(int argc, char** argv) {
   //First thing to do, activate the module !
   globus_module_activate(GLOBUS_IO_MODULE);

   globus_io_attr_t io_attr;
   globus_io_tcpattr_init(&io_attr);
```

```
gss_cred_id_t credential_handle = GSS_C_NO_CREDENTIAL;
if (!getCredential(&credential_handle)) {
   cerr << “you are not authenticated”;
   exit(1);
}

globus_io_secure_authorization_data_t auth_data;
globus_io_secure_authorization_data_initialize (&auth_data);
globus_result_t t;
_(globus_io_attr_set_secure_authentication_mode(
               &io_attr,
               GLOBUS_IO_SECURE_AUTHENTICATION_MODE_GSSAPI,  // use GSI
               credential_handle));
_(globus_io_attr_set_secure_authorization_mode(
     &io_attr,
     GLOBUS_IO_SECURE_AUTHORIZATION_MODE_SELF,
     &auth_data));
// We want encrypted communication !
// if not use GLOBUS_IO_SECURE_CHANNEL_MODE_CLEAR
_(globus_io_attr_set_secure_channel_mode(
      &io_attr,
      GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP));
_(globus_io_attr_set_secure_protection_mode(
      &io_attr,
      GLOBUS_IO_SECURE_PROTECTION_MODE_PRIVATE)); //encryption
// will see later for the delegation
_(globus_io_attr_set_secure_delegation_mode(
      &io_attr,
      GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY));
_(globus_io_attr_set_secure_proxy_mode(
                  &io_attr,
                  GLOBUS_IO_SECURE_PROXY_MODE_MANY));

// The first argument like http://m0.itso-tupi.com:10000 is
// parsed by using the globus_url_parse() function
globus_url_t     parsed_url;
if (globus_url_parse(argv[1], &parsed_url)!=GLOBUS_SUCCESS) {
   cerr << “invalid URL” << endl;
   exit(1);
};

globus_io_handle_t connection;
// use globus_io_tcp_register_connect for
// asynchronous connect. Here, this is a blocking call
_(globus_io_tcp_connect(
      parsed_url.host,
      parsed_url.port,
      &io_attr,
```

```
        &connection));
   cout << parsed_url.host << endl << parsed_url.port << endl;

   globus_size_t n;
   string msg("Hello world (secured) !");
   _(globus_io_write(&connection,
         (globus_byte_t*)STR(msg),
         msg.length(),
         &n));
   cout << n <<endl;
};
```

Example 7-13 globus-io example server - gsisocketserver.C

```
#include <iostream>
#include <globus_io.h>
#include <globus_io.h>
#include "globus_gss_assist.h"

//***************************************************************************
// This macro is defined for debugging reasons.  It checks the status of the
// globus calls and displays the Globus error message
// The t variable needs to be defined before
// ***************************************************************************
#define _(a) t=a;\
        if (t!=GLOBUS_SUCCESS) {\
                cerr <<
globus_object_printable_to_string(globus_error_get(t));\
                exit(1);\
        }
//***************************************************************************
// This function is used to check the validity of the local credentials
// probably generated by the grid-proxy-init
bool getCredential(gss_cred_id_t* credential_handle) {
           OM_uint32 major_status;
           OM_uint32 minor_status;

           major_status = globus_gss_assist_acquire_cred(&minor_status,
                                GSS_C_INITIATE, /* or GSS_C_ACCEPT */
                                     credential_handle);

           if (major_status != GSS_S_COMPLETE)
                   return false;
           else
                   return true;
}

//***************************************************************************
```

```
// Main program: create a listen socket, receive the message and close the
socket
// _() macro is used to check the error code of each Globus function and
// display the Globus Error message
// ********************************************************************************
main() {
   //First thing to do, activate the module !
    globus_module_activate(GLOBUS_IO_MODULE);
    globus_result_t t;

    globus_io_attr_t io_attr;
         globus_io_tcpattr_init(&io_attr);

    gss_cred_id_t credential_handle = GSS_C_NO_CREDENTIAL;
    // Authenticate with the GSSAPI library
        if (!getCredential(&credential_handle)) {
                cerr << “you are not authenticated”;
                exit(1);
        };

   globus_io_secure_authorization_data_t auth_data;
   globus_io_secure_authorization_data_initialize (&auth_data);
   _(globus_io_attr_set_secure_authentication_mode(
      &io_attr,
      GLOBUS_IO_SECURE_AUTHENTICATION_MODE_GSSAPI,
      credential_handle));
      _(globus_io_attr_set_secure_authorization_mode(
                &io_attr,
           GLOBUS_IO_SECURE_AUTHORIZATION_MODE_SELF,
                &auth_data));
      // We want encrypted communication !
      // if not use GLOBUS_IO_SECURE_CHANNEL_MODE_CLEAR
        _(globus_io_attr_set_secure_channel_mode(
                &io_attr,
                GLOBUS_IO_SECURE_CHANNEL_MODE_GSI_WRAP));
         _(globus_io_attr_set_secure_protection_mode(
                &io_attr,
                GLOBUS_IO_SECURE_PROTECTION_MODE_PRIVATE)); //encryption
        // will see later for the delegation
        _(globus_io_attr_set_secure_delegation_mode(
                &io_attr,
                GLOBUS_IO_SECURE_DELEGATION_MODE_FULL_PROXY));
   _(globus_io_attr_set_secure_proxy_mode(
         &io_attr,
         GLOBUS_IO_SECURE_PROXY_MODE_MANY));

   unsigned short port=10000;
   globus_io_handle_t handle;
   _(globus_io_tcp_create_listener(
```

```
            &port,
            -1,
            &io_attr,
            &handle));

    _(globus_io_tcp_listen(&handle));
    globus_io_handle_t newhandle;
    _(globus_io_tcp_accept(&handle,GLOBUS_NULL,&newhandle));
    globus_size_t n;
    char buf[500];
    _(globus_io_read( &newhandle,
            (globus_byte_t*)buf,
            500,
            5,
            &n));
    cout << buf << n << endl;
}
```

7.3 Global access to secondary storage

This section provides examples of using the GASS API.

7.3.1 Easy file transfer by using globus_gass_copy API

The Globus GASS Copy library is motivated by the desire to provide a uniform interface to transfer files via different protocols.

The goals in doing this are to:

- Provide a robust way to describe and apply file transfer properties for a variety of protocols: HTTP, FTP, and GSIFTP.
- Provide a service to support non-blocking file transfer and handle asynchronous file and network events.
- Provide a simple and portable way to implement file transfers.

The example in “ITSO_GASS_TRANSFER” on page 306 provides a complete implementation of a C++ class able to transfer files between two storage locations that could transparently be: A local file, a GASS server, a GridFTP server.

Note: The complete documentation for this API is available at:

http://www-unix.globus.org/api/c-globus-2.2/globus_gass_copy/html/index.html

The Globus Toolkit 2.2 uses a handle of type globus_gass_copy_handle_t to manage GASS copy. This handle is jointly used with three other specific handles that help to define the characteristics of the GASS operation:

- A handle of type globus_gass_copy_attr_t used for each remote location involved in the transfer (via gsiftp or http(s) protocol).
- A handle of type globus_gass_copy_handleattr_t used for the globus_gass_copy_handle_t initialization.
- A handle of type globus_gass_transfer_requestattr_t (request handle) used by the gass_transfer API to associate operations with a single file transfer request. It is used in the globus_gass_copy_attr_set_gass() call that specifies that we are using the GASS protocol for the transfer. This handle is also used by the gass_transfer API to determine protocol properties. In the example we specify binary transfer by calling globus_gass_transfer_requestattr_set_file_mode().

All these handlers need to be initialized before by using a `globus_gass_copy_*_init()` call specific to each handler.

The Globus Toolkit 2.2 provides the following functions to submit asynchronous transfers from an application:

- globus_gass_copy_register_handle_to_url() to copy a local file to a remote location.
- globus_gass_copy_register_url_to_handle() to copy a remote file locally.
- globus_gass_copy_register_url_to_url() to copy a remote file to a remote location.

This function uses a callback function that will be called when the transfer has been completed. The prototype of this function is defined by globus_gass_copy_callback_t type. A synchronization mechanism, like condition variables, must be used by the calling thread to be aware of the completion of the transfer. See Example 7-4 on page 180.

Globus Toolkit 2.2 provides blocking calls that are equivalent to those listed above:

- globus_gass_copy_handle_to_url() to copy a local file to a remote location
- globus_gass_copy_url_to_handle() to copy a remote file locally
- globus_gass_copy_url_to_url() to copy a remote file to a remote location

The globus_gass_copy_url_mode() function allows the caller to find out which protocol will be used for a given URL.

globus_url_parse() determines the validity of a URL.

Note: The GASS Transfer API is defined in the header file globus_gass_copy.h, and GLOBUS_GASS_COPY_MODULE must be activated before calling any functions in this API.

GAS Copy example

The best example is the `globus-url-copy.c` source code provided in the Globus Toolkit 2.2. It is strongly advised to have a look at this source code to understand how to use Globus Toolkit GASS calls.

This example shows how to copy a local file remotely via a GASS server. A GASS server needs to be started on the remote location. Note that this example is incomplete in the sense that it does not check any error codes returning from the Globus calls. Consequently, any malformed URL can cause the program to hang or fail miserably.

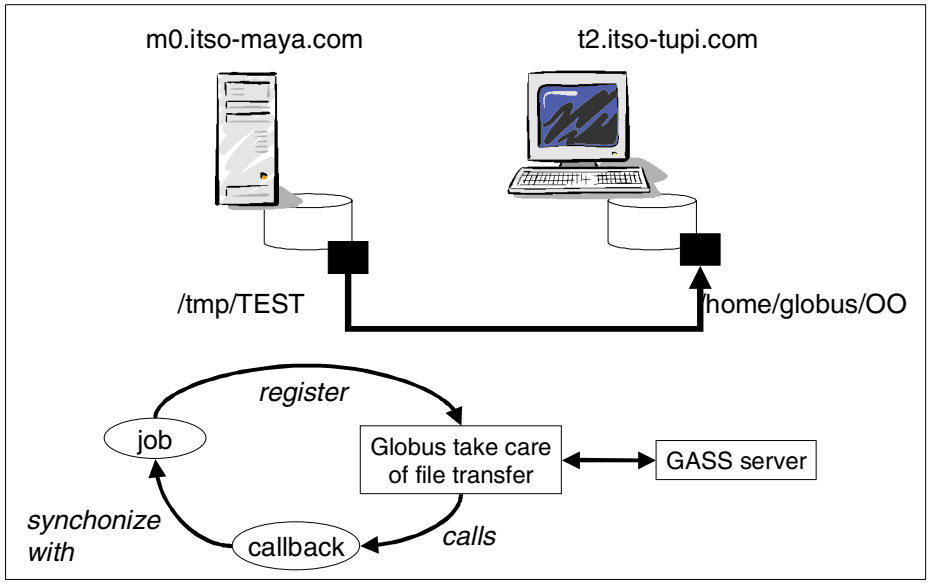


Figure 7-4 GASS Copy example

Example 7-14 gasscopy.C

```
#include “globus_common.h”
#include “globus_gass_copy.h”
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <cstring>
```

```
/*******************************************
 *  For a mote complete example
 *  see globus-url-copy.c
 *******************************************/

//*************************************************************************
//  GLOBUS_FILE is a class that acts as a wrapper to the globus_io_handle_t
//  globus_io_handle_t is taken as a parameter to
globus_gass_copy_register_handle_to_url
//  GLOBUS_URL is taken as a parameter to startTransfer() method of
//  the GASS_TRANSFER class
//*************************************************************************
class GLOBUS_FILE {
   globus_io_handle_t *io_handle;
   int       file_fd;
      public:
   GLOBUS_FILE() {};
   GLOBUS_FILE(char* filename) {
      io_handle =(globus_io_handle_t *)
                    globus_libc_malloc(sizeof(globus_io_handle_t));
      file_fd=open(filename,O_RDONLY);
       /* Globus function that converts a file descriptor
        *   into a globus_io_handle */
       globus_io_file_posix_convert(file_fd,
                                    GLOBUS_NULL,
                                    io_handle);
   };
   ~GLOBUS_FILE(){
      close(file_fd);
      globus_libc_free(io_handle);
   }
   globus_io_handle_t * get_globus_io_handle() {
      return io_handle;
   };
};

//*************************************************************************
//  GLOBUS_URL is a class that acts as a wrapper to the globus_url_t
//  globus_url_t is taken as a parameter to
//  globus_gass_copy_register_handle_to_url
//  GLOBUS_URL is taken as a parameter to startTransfer() method of
//  the GASS_TRANSFER class
//  setURL() allows to set up the URL as it is not set up in the constructor
//  globus_url_parse() is used to check the syntax of the url
//  globus_gass_copy_get_url_mode() determine the transfer mode
//    http,https,gsiftp of the url.  The type of this transfer mode
//    is globus_gass_copy_url_mode_t
//  getMode() returns this mode
//  getScheme() returns the scheme (http/https)
```

```
//  getURL() returns the string of the URL
//*************************************************************************
class GLOBUS_URL {
   globus_url_t url;
   globus_gass_copy_url_mode_t url_mode;
   char*        URL;
   public:
      GLOBUS_URL() {};
      ~GLOBUS_URL() {
         free(URL);
      };
      bool setURL(char* destURL) {
          //check if this is a valid URL
         if (globus_url_parse(destURL, &url) != GLOBUS_SUCCESS) {
                   cerr << "can not parse destURL" << destURL << endl;
              return false;
         }
         //determine the transfer mode
         if (globus_gass_copy_get_url_mode(destURL, &url_mode) !=
GLOBUS_SUCCESS) {
              cerr << "failed to determine mode for destURL" << destURL <<
endl;
              return false;
         };
         URL=strdup(destURL);
         return true;
      };
      globus_gass_copy_url_mode_t getMode() {
         return url_mode;
      };
      char* getScheme() {
         return url.scheme;
      }
      char* getURL() {
         return URL;
      }
};

//*************************************************************************
//  MONITOR implements the callback mechanism used in all the Globus
//  asynchronous mechanism: a non blocking globus call register an operation
//  and when this operation has been completed , this function is call.
//  To implement this mechanism in C++, we need to use a static function
//  that will a pointer to a MONITOR object as one argument.  This function
//  will then be able to call the callback method of this object (setDone()).
//
//  The Class ITSO_CB will be used in all other examples.  It is more complete
//  an easier to use but hide the details.
//
```

```
//  The callback implies a synchronization mechanism between the calling thread
//  and the callback.  To ensure thread safety and portability we use globus
//  function to manipulate the mutex the condition variable.
//
//  The class attributes are a mutex of type globus_mutex_t and a
//  condition variables of type globus_cond_t.  The C function globus_mutex_*
//  and globus_cond_* are used to manipulate them.  They maps the POSIX calls.
//
//  Other attributes are used to store information about the result of the ope
//  ration (done or error).
//
//  setDone() is called to indicate the operation has completed
//  (globus_gass_copy_register_handle_to_url).  It sends the signal via
//  the condition variable
//  Lock() and Unlock() locks and locks the mutex
//  Wait() waits the signal on the condition variable
//*************************************************************************
class MONITOR {
     globus_mutex_t                mutex;
          globus_cond_t                 cond;
          globus_object_t *             err;
     globus_bool_t                 use_err;
     globus_bool_t              done;
   public:
     MONITOR() {
      globus_mutex_init(&mutex, GLOBUS_NULL);
       globus_cond_init(&cond, GLOBUS_NULL);
      done = GLOBUS_FALSE;
      use_err = GLOBUS_FALSE;
     }
     ~MONITOR() {
               globus_mutex_destroy(&mutex);
               globus_cond_destroy(&mutex);
     };
     //-------------------
     void setError(globus_object_t* error) {
            use_err = GLOBUS_TRUE;
                     err = globus_object_copy(error);
     };
     //-------------------
     void setDone() {
         globus_mutex_lock(&mutex);
              done = GLOBUS_TRUE;
         globus_cond_signal(&cond);
         globus_mutex_unlock(&mutex);

     }
     //-------------------
     void Wait() {
```

```
            globus_cond_wait(&cond, &mutex);
      };
      //-------------------
      void Lock() {
       globus_mutex_lock(&mutex);
      };
      //-------------------
      void UnLock() {
       globus_mutex_unlock(&mutex);
      };
      //-------------------
      bool IsDone() {
         return done;
      };
};

//************************************************************
//  callback calls when the copy operation has finished
//  globus_gass_copy_register_handle_to_url() takes this function as
//  a parameter.  In C++, a class method cannot be passed as
//  a parameter to this function and we must use an intermediate
//  C funtion that will call this method.  Consequently
//  the object is used as the callbback argument so that
//  this C function knows which method it must call: monitor->setDone()
//************************************************************
static void
globus_l_url_copy_monitor_callback(
          void * callback_arg,
          globus_gass_copy_handle_t * handle,
          globus_object_t * error)
{
       MONITOR*              monitor;
            globus_bool_t         use_err = GLOBUS_FALSE;
       monitor = (MONITOR*)  callback_arg;

            if (error != GLOBUS_SUCCESS)
       {
            cerr << " url copy error:" <<
globus_object_printable_to_string(error) << endl;
            monitor->setError(error);
          }
       monitor->setDone();
        return;
} /* globus_l_url_copy_monitor_callback() */

//*********************************************************************
//  This Class implements the transfer from one local file to a GASS
//  url (http, https).  The class ITSO_GASS_TRANSFER implements a more
//  cmoplete set of transfer (source or destination can be either file,
```

```
//  http,https or gsiftp).  See appendix 2 for its source or
//  HelloWorld example for an example how to use it
//  To use it you must call setDestination() to register your destination
//  url.
//  setBinaryMode() wraps the globus_gass_transfer_requestattr_set_file_mode()
//  and is an example how to set up options that applies to the kind
//  of transfer.  These options are specific to the protocol
//  startTransfer() wraps the call to globus_gass_copy_register_handle_to_url()
//  that registers the asynchronous copy operation in the Globus API.  The
//  monitor object that manages the callback as well as the C function that
//  will call the callback object are passed as an arguement.
//**********************************************************************
class GASS_TRANSFER {
   globus_gass_copy_handle_t          gass_copy_handle;
   globus_gass_copy_handleattr_t      gass_copy_handleattr;
   globus_gass_transfer_requestattr_t*dest_gass_attr;
   globus_gass_copy_attr_t dest_gass_copy_attr;
   public:
   GASS_TRANSFER() {
      // handlers initialisation
      // first the attributes
      // then the gass copy handler
      globus_gass_copy_handleattr_init(&gass_copy_handleattr);
      globus_gass_copy_handle_init(&gass_copy_handle, &gass_copy_handleattr);
   };
   void setDestination(GLOBUS_URL& dest_url) {
        dest_gass_attr = (globus_gass_transfer_requestattr_t*)
          globus_libc_malloc (sizeof(globus_gass_transfer_requestattr_t));
        globus_gass_transfer_requestattr_init(dest_gass_attr,
                       dest_url.getScheme());
      // And We use GASS as transfer
      globus_gass_copy_attr_init(&dest_gass_copy_attr);
      globus_gass_copy_attr_set_gass(&dest_gass_copy_attr, dest_gass_attr);
   };
   void setBinaryMode() {
      globus_gass_transfer_requestattr_set_file_mode(
                dest_gass_attr,
                GLOBUS_GASS_TRANSFER_FILE_MODE_BINARY);
   };
   void startTransfer(GLOBUS_FILE& globus_source_file, GLOBUS_URL destURL,
MONITOR& monitor) {
   globus_result_t result = globus_gass_copy_register_handle_to_url(
            &gass_copy_handle,
                        globus_source_file.get_globus_io_handle(),
                    destURL.getURL(),
                   &dest_gass_copy_attr,
                    globus_l_url_copy_monitor_callback,
                    (void *) &monitor);
   };
```

```
};

//*************************************************************
//
//
//*************************************************************
main(int argc, char** argv) {

   char * localFile=strdup(argv[1]);
   char * destURL=strdup(argv[2]);
   cout << localFile << endl << destURL << endl;

   //Globus modules needs always to be activated
   // return code not checked here
   globus_module_activate(GLOBUS_GASS_COPY_MODULE);

   // Callback activation to monitor data transfer
   MONITOR monitor;


        /* convert file  into a globus_io_handle */
   GLOBUS_FILE globus_source_file(localFile);

    //check if this is a valid URL
   GLOBUS_URL dest_url;
   if (!dest_url.setURL(destURL))
      exit(2);

   // we do not manage gsiftp transfer ... not yet
   // see ITSO_GASS_TRANSFER for that or globus-url-copy.c
   if (dest_url.getMode() != GLOBUS_GASS_COPY_URL_MODE_GASS) {
      cerr << "You can only use GASS copy" << endl;
      exit(1);
   }

   GASS_TRANSFER transfer;
   transfer.setDestination(dest_url);

   //Use Binary mode !
   transfer.setBinaryMode();

   transfer.startTransfer(globus_source_file, dest_url, monitor);

   // Way to wait for a cond_signal by using a mutex and a condition
   // variable.  These three calls are included in the Wait() method
   // of the ITSO_CB call but they still use a mutex and condition
   // variable the same way.
   monitor.Lock();
   //wait until it is finished !
```

```
   while(!monitor.IsDone())
        {
            monitor.Wait();
   }
   monitor.UnLock();
};
```

To compile this example uses the following commands:

```
g++ -I /usr/local/globus/include/gcc32 -L/usr/local/globus/lib -o gasscopy
gasscopy.C -lglobus_gass_copy_gcc32 -lglobus_common_gcc32
```

To run the program, you need to start a GASS server on the remote site, for example:

```
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Sat Mar  1 02:40:36 2003
[globus@m0 globus]$ globus-gass-server -p 5000
https://m0.itso-maya.com:5000
```

On the client side, to copy the file /tmp/TEST to m0.itso-maya.com by renaming it to NEWTEST, issue:

```
[globus@t1 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Sat Mar  1 02:40:36 2003
[globus@t1 globus]$ ./gasscopy /tmp/TEST https://m0.itso-maya.com:5000/NEWTEST
```

On m0, you can check that NEWTEST appears in the target directory.

Note: If you use gsissh to connect from m0 to t1 after you issued grid-proxy-init, you do not need to reiterate grid-proxy-init because gsissh supports proxy delegation.

7.3.2 globus_gass_transfer API

The gass_transfer API is a core part of the GASS component of the Globus Toolkit. It provides a way to implement both client and server components.

- Client-specific functions are provided to implement file get, put, and append operations.
- Server-specific functions are provided to implement servers that service such requests.

The GASS Transfer API is easily extendable to support different remote data access protocols. The standard Globus distribution includes both client- and server-side support for the http and https protocols. An application that requires additional protocol support may add this through the protocol module interface.

globus_gass_transfer_request_t request handles are used by the gass_transfer API to associate operations with a single file transfer request.

The GASS transfer library provides both blocking and non-blocking versions of all its client functions.

7.3.3 Using the globus_gass_server_ez API

This API provides simple wrappers around the globus_gass_transfer API for server functionality. By using a simple function, globus_gass_server_ez_init(), you can start a GASS server that can perform the following functions:

- Write to local files with optional line buffering.
- Write to stdout and stderr.
- Shut down callback so the client can stop the server.

This API is used by the **globusrun** shell commands to embed a GASS server within it.

The example in "gassserver.C" on page 355 implements a simple GASS server and is an example of how to use this simple API.

The class ITSO_CB in "ITSO_CB" on page 315, and the function callback_c_function are used to implement the callback mechanism invoked when a client wants to shut down the GASS server. This mechanism is activated by setting the options GLOBUS_GASS_SERVER_EZ_CLIENT_SHUTDOWN_ENABLE when starting the GASS server.

The examples in "StartGASSServer() and StopGASSServer()" on page 324 provide two functions that wrap the Globus calls.

Example 7-15 Using ITSO_CB class as a callback for globus_gass_server_ez_init()

```
ITSO_CB callback; //invoked when client wants to shutdown the server

void callback_c_function() {
   callback.setDone();

main() {
......
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_CLIENT_SHUTDOWN_ENABLE;
.....
```

```
int err = globus_gass_server_ez_init(&listener,
                                     &attr,
                                     scheme,
                                     GLOBUS_NULL, //purpose unknown
                                     server_ez_opts,
                callback_c_function); //or GLOBUS_NULL otherwise
                //GLOBUS_NULL); //or GLOBUS_NULL otherwise
....
}
```

Various server options can be set, as shown in Example 7-16.

Example 7-16 Server options settings

```
// let s define options for our GASS server
unsigned long server_ez_opts=OUL;

//Files openfor writing will be written a line at a time
//so multiple writers can access them safely.
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_LINE_BUFFER;

//URLs that have ~ character, will be expanded to the home
//directory of the user who is running the server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_EXPAND;

//URLs that have ~user character, will be expanded to the home
//directory of the user on the server machine
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_USER_EXPAND;

//"get" requests will be fullfilled
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_READ_ENABLE;

//"put" requests will be fullfilled
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_WRITE_ENABLE;

// for put requets on /dev/stdout will be redirected to the standard
// output stream of the gass server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDOUT_ENABLE;

// for put requets on /dev/stderr will be redirected to the standard
// output stream of the gass server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDERR_ENABLE;

// "put requests" to the URL https://host/dev/globus_gass_client_shutdown
// will cause the callback function to be called.  this allows
// the GASS client to communicate shutdown requests to the server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_CLIENT_SHUTDOWN_ENABLE;
```

Before starting the server with globus_gass_server_ez_init() a listener must be created. This is the opportunity to:

- Define a port number on which the GASS server will listen.
- Select the protocol as secure or unsecure.

Example 7-17 Protocol selection or scheme

```
   // Secure
   char* scheme="https";
   //unsecure
   //char* scheme="http";
        globus_gass_transfer_listenerattr_t  attr;
        globus_gass_transfer_listenerattr_init(&attr, scheme);

        //we want to listen on post 10000
   globus_gass_transfer_listenerattr_set_port(&attr, 10000);
```

At this point the GASS server can be started. The GLOBUS_GASS_SERVER_EZ_MODULE must already be activated. The Wait() method of ITSO_CB uses a mutex/condition variable synchronization to ensure thread safety.

GASS server example

Below is a GASS server example.

Example 7-18 Starting GASS server

```
#include "globus_common.h"
#include "globus_gass_server_ez.h"
#include <iostream>
#include "itso_cb.h"

main() {
   // Never forget to activate GLOBUS module
   globus_module_activate(GLOBUS_GASS_SERVER_EZ_MODULE);

......

   //Now, we can start this gass server !
   globus_gass_transfer_listener_t    listener;
   globus_gass_transfer_requestattr_t  * reqattr      = GLOBUS_NULL; //purpose
unknown


   int err = globus_gass_server_ez_init(&listener,
                &attr,
                scheme,
```

```
                GLOBUS_NULL, //purpose unknown
                server_ez_opts,
                callback_c_function); //or GLOBUS_NULL otherwise

   if((err != GLOBUS_SUCCESS)) {
           cerr << "Error: initializing GASS (" << err << ")" << endl;
          exit(1);
   }
char *  gass_server_url=globus_gass_transfer_listener_get_base_url(listener);
   cout << "we are listening on " << gass_server_url << endl;

   //wait until it is finished !
/  /that means that the "put requests" to the URL
//https://host/dev/globus_gass_client_shutdown
   //ITSO_CB implements the symchronization mechanism by using a mutex
   //and a condition variable
        callback.Wait();  // shutdown callback

   //stop everything
   globus_gass_server_ez_shutdown(listener);
   globus_module_deactivate(GLOBUS_GASS_SERVER_EZ_MODULE);
```

To compile this program issue, use the following Makefile:

```
#globus-makefile-header --flavor gcc32 globus_gass_server_ez globus_common
globus_gass_transfer globus_io globus_gass_copy > globus_header

include globus_header

all:  gassserver

%.o: %.C
   g++ -g -c $(GLOBUS_CPPFLAGS) $< -o $@

gassserver: gassserver.o itso_cb.o
   g++ -g -o $@ $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS)
```

This program can be launched on one node (for example, m0.itso.maya.com), and by using gasscopy from another node (for example, t2.itso-tupi.com), we will be able to copy files, display files on m0, and even shut down the GASS server.

On m0.itso-maya.com:

```
./gassserver
```

On t2.itso-tupi.com:

```
./gasscopy FileToBeCopied https://m0.itso-maya.com:10000/FileCopied
./gasscopy FileToBeDisplayed https://m0.itso-maya.com:10000/dev/stdout
./gasscopy None https://m0.itso-maya.com:10000/dev/globus_gass_client_shutdown
```

Note: On both server and client side, you need to have the same credentials. This is achieved when you submit a job via the gatekeeper that supports proxy delegation or using gsissh.

7.3.4 Using the globus-gass-server command

globus-gass-server is a simple file server that can be used by any user when necessary from a Unix shell. It uses the secure https protocol and GSI security infrastructure.

The GASS server can be started with or without GSI security. The security mode is controlled by the -i option that deactivates the GSSAPI security. This way the server will use http protocol instead of https protocol.

The -c option allows a client to shut down the server by writing to dev/globus_gass_client_shutdown. See the previous example.

The -o and -e options allow a client to write to standard output and standard error.

The -r and -w options authorize a client to respectively read and write on the local file system where the GASS server is running.

The -t option expands the tilde sign (~) in a URL expression to value of the user's home directory.

globus-gass-server example

On m0.itso-maya.com:

[globus@m0 globus]globus-gass-server -o -e -r -w p 10001

On t2.itso-tupi.com:

```
[globus@m0 globus]globus-url-copy file:///home/globus/FileToBeCopied
https://m0.itso.maya.com:10001/dev/stdout
```

You can see the contents of the FileToBeCopied file on m0.

7.3.5 Globus cache management

The globus-gass-cache API provides an interface for management of the GASS cache.

globus-gass-cache

The Globus Toolkit 2.2 provides command line tools (globus-gass-cache), as well as a C API that can be used to perform operations on the cache. The operations are:

- **`add`**: Add the URL to the cache if it is not there. If the URL is already in the cache, increment its reference count.
- **`delete`**: Remove a reference to the URL with this tag; if there are no more references, remove the cached file.
- **`cleanup-tag`**: Remove all references to this tag for the specified URL, or all URLs (if no URL is specified).
- **`cleanup-url`**: Remove all tag references to this URL.
- **`list`**: List the contents of the cache.

The GASS cache is used when a job is submitted via the GRAM sub-system. The count entry in the RSL parameters allows control of how long the program will stay in the cache. When forgotten, the file will remain forever. A common problem is to rerun a program in the cache after you have modified it locally:

```
&(executable=https://m0.itso-maya.com:20000/home/globus/Compile)
```

On the execution host, the binary will be tagged as https://m0.itso-maya.com:20000/home/globus/Compile. If modified on m0, it will not be modified on the cache. Consequently, the wrong program will be run on m0. You can check the cache on the remote server with globus-gass-cache -list. Use globus-gass-cache -clean-up to remove all the entries in the cache. The way to avoid this problem is to use (count=1) in the RSL commands. Count specifies that you only want to run the executable once.

Below is a set of examples to illustrate cache management using Globus Toolkit shell commands.

Example 7-19 shows how to create a copy on t2.itso-tupi.com of the file gsiclient2 stored on a GSIFTP server at t0.itso-tupi.com and request the file. The file will be referred to with the tag itso.

Example 7-19 Adding a file to the cache

```
globus-gass-cache -add -t itso -r t2 gsiftp://t0/home/globus/gsiclient2
```

The file is not stored in the cache with the same file name. Use the **`globus-gass-cache`** command to retrieve the file, as shown in Example 7-20.

Example 7-20 Retrieving a file in the cache

```
globus-gass-cache -list -r t2
```

```
URL: gsiftp://t0//home/globus/gsiclient2
        Tag:itso
globus-gass-cache -query -t itso -r t2 gsiftp://t0//home/globus/gsiclient2
```

It returns the name of the file in the cache.

```
/home/globus/.globus/.gass_cache/local/md5/4e/72/68/e57a109668e83f60927154d812/
md5/a6/78/0e/703376a3006db586eb24535315/data
```

You can then invoke it using **globusrun**, as shown in Example 7-21.

Example 7-21 Invoking a program from the cache

```
globusrun -o -r t2
'&(executable=/home/globus/.globus/.gass_cache/local/md5/4e/72/68/e57a109668e83
f60927154d812/md5/a6/78/0e/703376a3006db586eb24535315/data)
(arguments=https://g0.itso-tupi.com:10000)'
```

Files in the cache are usually referenced with a tag equal to a URL. You can use the file name or the tag to remove the file from the cache. GASS refers to the files in the cache with a tag equal to their URL.

The following command removes a single reference of tag itso from the specified URL. If this is the only tag, then the file corresponding to the URL on the local machine's cache will be removed.

```
globus-gass-cache -delete -t itso gsiftp://t0//home/globus/gsiclient2
```

The following removes a reference to the tag itso from all URLs in the local cache:

```
globus-gass-cache -cleanup-tag -t itso
```

To remove all tags for the URL gsiftp://t0//home/globus/gsiclient2, and remove the cached copy of that file:

```
globus-gass-cache -cleanup-tag gsiftp://t0//home/globus/gsiclient2
```

Note: $GRAM_JOB_CONTACT is the tag used for a job started from GRAM and that uses GASS. All $GRAM_JOB_CONTACT tags are deleted when the GRAM job manager completes.

7.4 GridFTP

The Globus Toolkit 2.2 uses an efficient and robust protocol for data movement. This protocol should be used whenever large files are involved instead of the http and https protocols that can also be used with the GASS subsystem.

The Globus Toolkit 2.2 provides a GridFTP server based on wu-ftpd code and a C API that can be used by applications to use GridFTP functionality. This GridFTP server does not implement all the features of the GridFTP protocol. It works only as a non-striped server even if it can inter-operate with other striped servers.

All Globus Toolkit 2.2 shell commands can transparently use the GridFTP protocol whenever the URL used for a file begins with gsiftp://.

7.4.1 GridFTP examples

The following example copies the jndi file located on m0.itso-maya.com to the host g2.itso-guarani.com. Note that this command can be issued on a third machine, such as t2.itso-tupi.com.

```
globus-url-copy gsiftp://m0/~/jndi-1_2_1.zip gsiftp://g2/~/jndi-1_2_1.zip
```

The following example executes on g2.itso-guarini.com a binary that is retrieved from g1.itso-guarani.com. This command could be issued from t3.itso-tupi.com.

```
globus-job-run g2 gsiftp://g1/bin/hostname
```

A grid-enabled application needs to use the GridFTP API to be able to transparently use Globus Toolkit 2 data grid features. This API is detailed in Globus GridFTP APIs.

7.4.2 Globus GridFTP APIs

This section discusses the APIs that can be used with GridFTP.

Skeletons for C/C++ applications

globus_module_activate(GLOBUS_FTP_CLIENT_MODULE) must be called at the beginning of the program to activate the globus_ftp_client module.

Within the globus_ftp_client API, all FTP operations require a handle parameter. Only one FTP operation may be in progress at once per FTP handle. The type of this handle is globus_ftp_client_handle_t, and must be initialized using globus_ftp_client_handle_init().

The properties of the FTP connection can be configured using another handle of type globus_ftp_client_handleattr_t that also must be initialized by using globus_ftp_client_handleattr_init().

By using these two handles, a client can easily execute all of the usual FTP commands:

- **globus_ftp_client_put(), globus_ftp_client_get(), globus_ftp_client_mkdir()**, **globus_ftp_client_rmdir()**, **globus_ftp_client_list(), globus_ftp_client_delete()**, **globus_ftp_client_verbose_list(), globus_ftp_client_move()**.
- **globus_ftp_client_exists()** tests the existence of a file of a directory.
- **globus_ftp_client_modification_time()** returns the modification time of a file.
- **globus_ftp_client_size()** returns the size of the file.

The globus_ftp_client*get() functions only start a get file transfer from an FTP server. If this function returns GLOBUS_SUCCESS, then the user may immediately begin calling globus_ftp_client_read() to retrieve the data associated with this URL.

Similarly, the globus_ftp_client*put() functions only start a put file transfer from an FTP server. If this function returns GLOBUS_SUCCESS, then the user may immediately begin calling globus_ftp_client_write() to write the data associated with this URL.

Example 7-22 First example extracted from the Globus tutorial

```
/*****************************************************************************
 * Globus Developers Tutorial: GridFTP Example - Simple Authenticated Put
 *
 * There are no handle or operation attributes used in this example.
 * This means the transfer runs using all defaults, which implies standard
 * FTP stream mode.  Note that while this program shows proper usage of
 * the Globus GridFTP client library functions, it is not an example of
 * proper coding style.  Much error checking has been left out and other
 * simplifications made to keep the program simple.
 *****************************************************************************/

#include <stdio.h>
#include "globus_ftp_client.h"


static globus_mutex_t lock;
static globus_cond_t cond;
static globus_bool_t done;

#define MAX_BUFFER_SIZE 2048
#define ERROR  -1
#define SUCCESS 0

/*****************************************************************
 * done_cb:  A pointer to this function is passed to the call to
 * globus_ftp_client_put (and all the other high level transfer
```

```
 * operations).   It is called when the transfer is completely
 * finished, i.e. both the data channel and control channel exchange.
 * Here it simply sets a global variable (done) to true so the main
 * program will exit the while loop.
 ********************************************************************/
static
void
done_cb(
        void *                                  user_arg,
        globus_ftp_client_handle_t *            handle,
        globus_object_t *                       err)
{
    char * tmpstr;

    if(err)
    {
        fprintf(stderr, "%s", globus_object_printable_to_string(err));
    }
    globus_mutex_lock(&lock);
    done = GLOBUS_TRUE;
    globus_cond_signal(&cond);
    globus_mutex_unlock(&lock);
    return;
}

/*********************************************************************
 * data_cb: A pointer to this function is passed to the call to
 * globus_ftp_client_register_write.  It is called when the user supplied
 * buffer has been successfully transferred to the kernel.  Note that does
 * not mean it has been successfully transmitted.  In this simple version,
 * it justs reads the next block of data and calls register_write again.
 *********************************************************************/
static
void
data_cb(
    void *                                  user_arg,
    globus_ftp_client_handle_t *            handle,
    globus_object_t *                       err,
    globus_byte_t *                         buffer,
    globus_size_t                           length,
    globus_off_t                            offset,
    globus_bool_t                           eof)
{
    if(err)
    {
        fprintf(stderr, "%s", globus_object_printable_to_string(err));
    }
    else
    {
```

```
        if(!eof)
        {
            FILE *fd = (FILE *) user_arg;
            int rc;
            rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
            if (ferror(fd) != SUCCESS)
            {
                printf("Read error in function data_cb; errno = %d\n", errno);
                return;
            }
            globus_ftp_client_register_write(
                handle,
                buffer,
                rc,
                offset + length,
                feof(fd) != SUCCESS,
                data_cb,
                (void *) fd);
        } /* if(!eof) */
    } /* else */
    return;
} /* data_cb */

/*************************
 * Main Program
 ************************/

int main(int argc, char **argv)
{
    globus_ftp_client_handle_t              handle;
    globus_byte_t                           buffer[MAX_BUFFER_SIZE];
    globus_size_t                           buffer_length = MAX_BUFFER_SIZE;
    globus_result_t                         result;
    char *                                  src;
    char *                                  dst;
    FILE *                                  fd;


    /***********************************
     * Process the command line arguments
     ***********************************/

    if (argc != 3)
    {
        printf("Usage: put local_file DST_URL\n");
        return(ERROR);
    }
    else
    {
```

```
    src = argv[1];
    dst = argv[2];
}

/*******************************
 * Open the local source file
 *******************************/
fd = fopen(src,"r");
if(fd == NULL)
{
    printf("Error opening local file: %s\n",src);
    return(ERROR);

}

/*********************************************************************
 * Initialize the module, and client handle
 * This has to be done EVERY time you use the client library
 * The mutex and cond are theoretically optional, but highly recommended
 * because they will make the code work correctly in a threaded build.
 *
 * NOTE: It is possible for each of the initialization calls below to
 * fail and we should be checking for errors.  To keep the code simple
 * and clean we are not.  See the error checking after the call to
 * globus_ftp_client_put for an example of how to handle errors in
 * the client library.
 *********************************************************************/

globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
globus_mutex_init(&lock, GLOBUS_NULL);
globus_cond_init(&cond, GLOBUS_NULL);
globus_ftp_client_handle_init(&handle,  GLOBUS_NULL);

/*********************************************************************
 * globus_ftp_client_put starts the protocol exchange on the control
 * channel.  Note that this does NOT start moving data over the data
 * channel
 *********************************************************************/
done = GLOBUS_FALSE;

result = globus_ftp_client_put(&handle,
                               dst,
                               GLOBUS_NULL,
                               GLOBUS_NULL,
                               done_cb,
                               0);
if(result != GLOBUS_SUCCESS)
{
    globus_object_t * err;
```

```
    err = globus_error_get(result);
    fprintf(stderr, "%s", globus_object_printable_to_string(err));
    done = GLOBUS_TRUE;
}
else
{
    int rc;

    /*************************************************************
     * This is where the data movement over the data channel is initiated.
     * You read a buffer, and call register_write.  This is an asynch
     * call which returns immediately.  When it is finished writing
     * the buffer, it calls the data callback (defined above) which
     * reads another buffer and calls register_write again.
     * The data callback will also indicate when you have hit eof
     * Note that eof on the data channel does not mean the control
     * channel protocol exchange is complete.  This is indicated by
     * the done callback being called.
     *************************************************************/
    rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
    globus_ftp_client_register_write(
        &handle,
        buffer,
        rc,
        0,
        feof(fd) != SUCCESS,
        data_cb,
        (void *) fd);
}

/*********************************************************************
 * The following is a standard thread construct.  The while loop is
 * required because pthreads may wake up arbitrarily.  In non-threaded
 * code, cond_wait becomes globus_poll and it sits in a loop using
 * CPU to wait for the callback.  In a threaded build, cond_wait would
 * put the thread to sleep
 *********************************************************************/
globus_mutex_lock(&lock);
while(!done)
{
    globus_cond_wait(&cond, &lock);
}
globus_mutex_unlock(&lock);
/*********************************************************************
 * Since done has been set to true, the done callback has been called.
 * The transfer is now completely finished (both control channel and
 * data channel).  Now, Clean up and go home
 *********************************************************************/
```

```
    globus_ftp_client_handle_destroy(&handle);
    globus_module_deactivate_all();

    return 0;
}
```

To compile the program:

```
gcc -I /usr/local/globus/include/gcc32 -L/usr/local/globus/lib -o
gridftpclient2 gridftpclient1.c -lglobus_ftp_client_gcc32
```

To use it:

```
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ............................................. Done
Your proxy is valid until: Thu Mar  6 02:17:53 2003

[globus@m0 globus]$ ./gridftpclient1 LocalFile gsiftp://g2/tmp/RemoteFile
```

Partial transfer

All operations are asynchronous and require a callback function that will be called when the operation has been completed. Mutex and condition variables must be used to ensure thread safety.

GridFTP supports partial transfer. To do this, you need to use offsets that will determine the beginning and the end of data that you want to transfer. The type of the offset is globus_off_t.

The globus_ftp_client_partial_put() and globus_ftp_client_partial_get() are used to execute the partial transfer.

The Globus FTP Client library provides the ability to start a file transfer from a known location in the file. This is accomplished by passing a restart marker to globus_ftp_client_get() and globus_ftp_client_put(). The type of this restart marker is globus_ftp_client_restart_marker_t and must be initialized by calling globus_ftp_client_restart_marker_init().

For a complete description of the globus_ftp_client API, see:

http://www--unix.globus.org/api/c/globus_ftp_client/html/index.html

Parallelism

GridFTP supports two kind of transfers:

- Stream mode is a file transfer mode where all data is sent over a single TCP socket, without any data framing. In stream mode, data will arrive in sequential order. This mode is supported by nearly all FTP servers.
- Extended block mode is a file transfer mode where data can be sent over multiple parallel connections and to multiple data storage nodes to provide a high-performance data transfer. In extended block mode, data may arrive out of order. ASCII type files are not supported in extended block mode.

Use globus_ftp_client_operationattr_set_mode() to select the mode. Note that you will need a control handler of type globus_ftp_client_operationattr_t to define this transfer mode, and it needs to be initialized before being used by the function globus_ftp_client_operationattr_init().

Currently, only a "fixed" parallelism level is supported. This is interpreted by the FTP server as the number of parallel data connections to be allowed for each stripe of data. Use the globus_ftp_client_operationattr_set_parallelism() to set up the parallelism.

You also need to define a layout that defines what regions of a file will be stored on each stripe of a multiple-striped FTP server. You can do this by using the function globus_ftp_client_operationattr_set_layout().

Example 7-23 Parallel transfer example extracted from Globus tutorial

```
/*****************************************************************************
 * Globus Developers Tutorial: GridFTP Example - Authenticated Put w/ attrs
 *
 * Operation attributes are used in this example to set a parallelism of 4.
 * This means the transfer must run in extended block mode MODE E.
 * Note that while this program shows proper usage of
 * the Globus GridFTP client library functions, it is not an example of
 * proper coding style.  Much error checking has been left out and other
 * simplifications made to keep the program simple.
 *****************************************************************************/

#include <stdio.h>
#include "globus_ftp_client.h"

static globus_mutex_t lock;
static globus_cond_t cond;
static globus_bool_t done;
int           global_offset = 0;

#define MAX_BUFFER_SIZE (64*1024)
#define ERROR  -1
```

```
#define SUCCESS 0
#define PARALLELISM 4

/************************************************************************
 * done_cb:  A pointer to this function is passed to the call to
 * globus_ftp_client_put (and all the other high level transfer
 * operations).   It is called when the transfer is completely
 * finished, i.e. both the data channel and control channel exchange.
 * Here it simply sets a global variable (done) to true so the main
 * program will exit the while loop.
 ************************************************************************/
static
void
done_cb(
        void *                                  user_arg,
        globus_ftp_client_handle_t *            handle,
        globus_object_t *                       err)
{
    char * tmpstr;

    if(err)
    {
        fprintf(stderr, "%s", globus_object_printable_to_string(err));
    }
    globus_mutex_lock(&lock);
    done = GLOBUS_TRUE;
    globus_cond_signal(&cond);
    globus_mutex_unlock(&lock);
    return;
}

/************************************************************************
 * data_cb: A pointer to this function is passed to the call to
 * globus_ftp_client_register_write.  It is called when the user supplied
 * buffer has been successfully transferred to the kernel.  Note that does
 * not mean it has been successfully transmitted.  In this simple version,
 * it justs reads the next block of data and calls register_write again.
 ************************************************************************/
static
void
data_cb(
    void *                                  user_arg,
    globus_ftp_client_handle_t *            handle,
    globus_object_t *                       err,
    globus_byte_t *                         buffer,
    globus_size_t                           length,
    globus_off_t                            offset,
    globus_bool_t                           eof)
{
```

```
    if(err)
    {
        fprintf(stderr, "%s", globus_object_printable_to_string(err));
    }
    else
    {
        if(!eof)
        {
            FILE *fd = (FILE *) user_arg;
            int rc;
            rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
            if (ferror(fd) != SUCCESS)
            {
                printf("Read error in function data_cb; errno = %d\n", errno);
                return;
            }
            globus_ftp_client_register_write(
                handle,
                buffer,
                rc,
                global_offset,
                feof(fd) != SUCCESS,
                data_cb,
                (void *) fd);
            global_offset += rc;
        } /* if(!eof) */
        else
        {
            globus_libc_free(buffer);
        }

    } /* else */
    return;
} /* data_cb */

/*************************
 * Main Program
 ************************/

int main(int argc, char **argv)
{
    globus_ftp_client_handle_t              handle;
    globus_ftp_client_operationattr_t       attr;
    globus_ftp_client_handleattr_t          handle_attr;
    globus_byte_t *                         buffer;
    globus_result_t                         result;
    char *                                  src;
    char *                                  dst;
    FILE *                                  fd;
```

```
globus_ftp_control_parallelism_t        parallelism;
globus_ftp_control_layout_t             layout;
int                                     i;

/************************************
 * Process the command line arguments
 ************************************/

if (argc != 3)
{
    printf("Usage: ext-put local_file DST_URL\n");
    return(ERROR);
}
else
{
    src = argv[1];
    dst = argv[2];
}

/*******************************
 * Open the local source file
 *******************************/
fd = fopen(src,"r");
if(fd == NULL)
{
    printf("Error opening local file: %s\n",src);
    return(ERROR);
}

/*********************************************************************
 * Initialize the module, handleattr, operationattr, and client handle
 * This has to be done EVERY time you use the client library
 * (if you don't use attrs, you don't need to initialize them and can
 * pass NULL in the parameter list)
 * The mutex and cond are theoretically optional, but highly recommended
 * because they will make the code work correctly in a threaded build.
 *
 * NOTE: It is possible for each of the initialization calls below to
 * fail and we should be checking for errors.  To keep the code simple
 * and clean we are not.  See the error checking after the call to
 * globus_ftp_client_put for an example of how to handle errors in
 * the client library.
 *********************************************************************/

globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
globus_mutex_init(&lock, GLOBUS_NULL);
globus_cond_init(&cond, GLOBUS_NULL);
globus_ftp_client_handleattr_init(&handle_attr);
globus_ftp_client_operationattr_init(&attr);
```

```
/*************************************************************************
 * Set any desired attributes, in this case we are using parallel streams
 ************************************************************************/

parallelism.mode = GLOBUS_FTP_CONTROL_PARALLELISM_FIXED;
parallelism.fixed.size = PARALLELISM;
layout.mode = GLOBUS_FTP_CONTROL_STRIPING_BLOCKED_ROUND_ROBIN;
layout.round_robin.block_size = 64*1024;
globus_ftp_client_operationattr_set_mode(
    &attr,
    GLOBUS_FTP_CONTROL_MODE_EXTENDED_BLOCK);
globus_ftp_client_operationattr_set_parallelism(&attr,
                                                &parallelism);

globus_ftp_client_operationattr_set_layout(&attr,
                                           &layout);

globus_ftp_client_handle_init(&handle,  &handle_attr);

/*************************************************************************
 * globus_ftp_client_put starts the protocol exchange on the control
 * channel.  Note that this does NOT start moving data over the data
 * channel
 ************************************************************************/
done = GLOBUS_FALSE;

result = globus_ftp_client_put(&handle,
                               dst,
                               &attr,
                               GLOBUS_NULL,
                               done_cb,
                               0);
if(result != GLOBUS_SUCCESS)
{
    globus_object_t * err;
    err = globus_error_get(result);
    fprintf(stderr, "%s", globus_object_printable_to_string(err));
    done = GLOBUS_TRUE;
}
else
{
    int rc;

    /*********************************************************
     * This is where the data movement over the data channel is initiated.
     * You read a buffer, and call register_write.  This is an asynch
     * call which returns immediately.  When it is finished writing
     * the buffer, it calls the data callback (defined above) which
```

```
     * reads another buffer and calls register_write again.
     * The data callback will also indicate when you have hit eof
     * Note that eof on the data channel does not mean the control
     * channel protocol exchange is complete.  This is indicated by
     * the done callback being called.
     *
     * NOTE: The for loop is present BECAUSE of the parallelism, but
     * it is not CAUSING the parallelism.  The parallelism is hidden
     * inside the client library.  This for loop simply insures that
     * we have sufficient buffers queued up so that we don't have
     * TCP steams sitting idle.
     ************************************************************/
    for (i = 0; i< 2 * PARALLELISM && feof(fd) == SUCCESS; i++)
    {
        buffer = malloc(MAX_BUFFER_SIZE);
        rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
        globus_ftp_client_register_write(
            &handle,
            buffer,
            rc,
            global_offset,
            feof(fd) != SUCCESS,
            data_cb,
            (void *) fd);
        global_offset += rc;

    }

}

/*****************************************************************
 * The following is a standard thread construct.  The while loop is
 * required because pthreads may wake up arbitrarily.  In non-threaded
 * code, cond_wait becomes globus_poll and it sits in a loop using
 * CPU to wait for the callback.  In a threaded build, cond_wait would
 * put the thread to sleep
 *****************************************************************/
globus_mutex_lock(&lock);
while(!done)
{
    globus_cond_wait(&cond, &lock);
}
globus_mutex_unlock(&lock);

/*****************************************************************
 * Since done has been set to true, the done callback has been called.
 * The transfer is now completely finished (both control channel and
 * data channel).  Now, Clean up and go home
 *****************************************************************/
```

```
    globus_ftp_client_handle_destroy(&handle);
    globus_module_deactivate_all();

    return 0;
}
```

To compile the program:

```
gcc -I /usr/local/globus/include/gcc32 -L/usr/local/globus/lib -o
gridftpclient2 gridftpclient2.c -lglobus_ftp_client_gcc32
```

To use it:

```
[globus@m0 globus]$ grid-proxy-init
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ............................................. Done
Your proxy is valid until: Thu Mar  6 02:17:53 2003
[globus@m0 globus]$ ./gridftpclient2 LocalFile gsiftp://g2/tmp/RemoteFile
```

Shells tools

globus-url-copy is the shell tool to use to transfer files from one location to another. It takes two parameters that are the URLs for the specific file. The prefix gsiftp://<hostname>/ is used to specify a GridFTP server.

The following example copies a file from the host m0 to the server a1:

```
globus-url-copy gsiftp://m0/tmp/FILE gsiftp://a1/~/tmp
```

The following example uses a GASS server started on host b0 and listening on port 23213:

```
globus-url-copy https://b0:23213/home/globus/OtherFile gsiftp://a1/~/tmp
```

The following example uses a local file as a source file:

```
globus-url-copy file:///tmp/FILE gsiftp://a1/~/tmp
```

7.5 Replication

To utilize replication, a replication server needs to be installed. It consists of an LDAP server. The Globus Toolkit 2.2 provides an LDAP server than can be used for this purpose. See “Installation” on page 211. In the Globus Toolkit 2.2 the GSI security infrastructure is not used to modify entries in the LDAP repository. Consequently, a password and an LDAP administrator need to be defined for the replica server. It will be used each time from the client side to perform write operations to the LDAP tree.

7.5.1 Shell commands

The Globus Toolkit 2.2 provides a single shell command for manipulating replica catalog objects. The format of the command is:

```
globus-replica-catalog HOST OBJECT ACTION
```

Where:

- HOST specifies the logical collection in the replica catalog as well of the information needed to connect to the LDAP server (a user and a password). The Globus Toolkit V2.2 uses an LDAP directory so the URL for a collection follows the format ldap://host:[port]/*dn* where *dn* is the distinguished name of the collection. The HOST format is therefore:

  -host <collection URL> -manager <manager DN> -password <file>

  Two environment variables can be used to avoid typing the -host and -manager option each time:

  - GLOBUS_REPLICA_CATALOG_HOST for the logical collection distinguished name.
  - GLOBUS_REPLICA_CATALOG_MANAGER for the manager distinguished name.
  - file contains the password used during the connection.
- OBJECT indicates which entry in the replica catalog the command will act upon:
  - -collection for a collection that was specified in the -host option
  - -location <name>
  - -logicalfile <name>
- ACTION determines which operations will be executed on the entry. There are four categories: Creation/deletion, attributes modifications, files names manipulation in the logical collection file lists and location file lists, and finally search operations. See the Globus documentation for more information.

7.5.2 Replica example

In the following example scenario, we propose to create a logical collection called itsoCollection in the Replica Catalog created in “Installation” on page 211. This collection consists of five files that are located on two different servers, g0.itso-guarani.com and t0.itso-tupi.com. Three files are stored on g0.itso-guarani.com, and two others are located on t0.itso-tupi.com. The two locations host a GridFTP server.

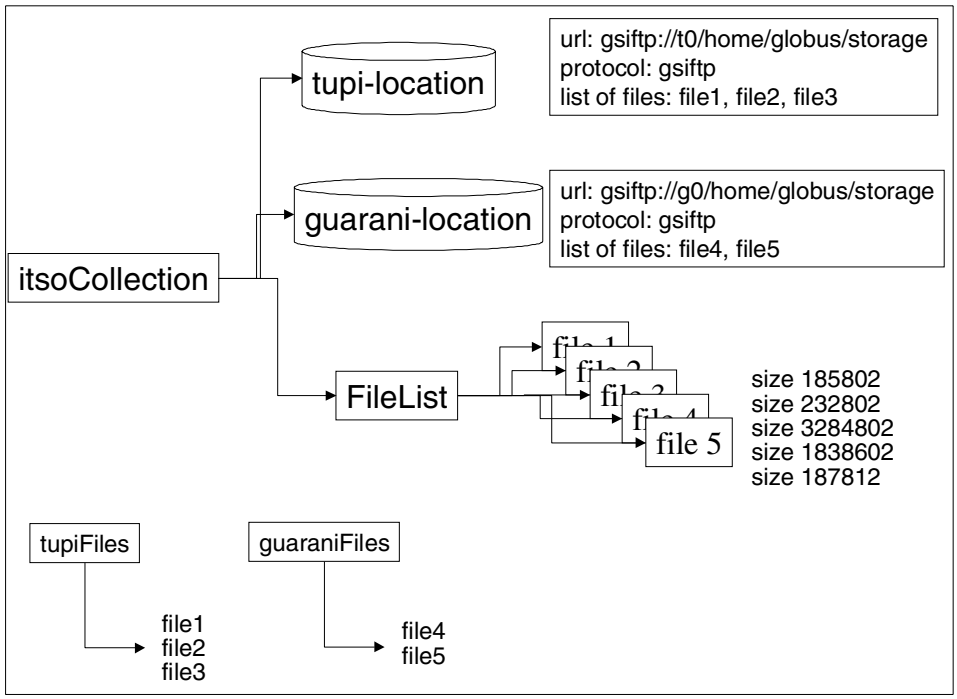


Figure 7-5 Replica example

The steps are:

1. First, set up the environment:

```
export GLOBUS_REPLICA_CATALOG_HOST="ldap://m0.itso-maya.com/lc=itso\
Collection,rc=test,dc=itso-maya,dc=com"
export GLOBUS_REPLICA_CATALOG_MANAGER="cn=Manager,dc=itso-maya,dc=com"
echo ###### > password
```

2. Create the three file lists: One for the files in the collection, one for the files located in g0.itso-guarani.com, and the last for the files stored on t0.itso-tupi.com.

```
for i in file1 file2 file3 file4 file5;do echo $i >> FileList;done
for i in file1 file2 file3 ;do echo $i >> tupiFiles;done
for i in file4 file5;do echo $i >> guaraniFiles;done
```

3. Register the collection:

```
globus-replica-catalog -password password -collection -create FileList
```

4. Register the two locations and their file list:

```
globus-replica-catalog -password password -location "t0 Tupi Storage"
-create "gridftp://t0.itso-tupi.com/home/globus/storage/" tupiFiles
```

```
globus-replica-catalog -password password -location “g0 Guarani Storage”
-create “gridftp://g0.itso-tupi.com/home/globus/storage/” guaraniFiles
```

5. Register each of the logical files with their size:

```
globus-replica-catalog -password password -logicalfile “file1” -create
100000
globus-replica-catalog -password password -logicalfile “file2” -create
200000
globus-replica-catalog -password password -logicalfile “file3” -create
300000
globus-replica-catalog -password password -logicalfile “file4” -create
400000
globus-replica-catalog -password password -logicalfile “file5” -create
500000
```

We can now perform a few requests:

1. Search for all locations that contain file4 and file5:
   a. Create a file FilesToBeFound that contains the files we are looking for:

```
for i in file4 file5; do echo $i >> FilesToBeFound;done
```

   b. Perform the request:

```
globus-replica-catalog -password password -collection -find-locations\
FilesToBeFound uc
```

   Then you should receive the following output:

```
filename=file4
filename=file5
uc=gridftp://g0.itso-tupi.com/home/globus/storage
```

   uc means URL Constructor and is the attribute used in the LDAP directory to store the location URL.

2. Check the size attribute for the file file2:

```
globus-replica-catalog -password password -logicalfile “file2”\
-list-attributes size
```

   You receive:

```
size=200000
```

7.5.3 Installation

The installation process is explained at:

http://www.globus.org/gt2/replica.html

It consists of the following steps:

1. Add a new schema that defines objects manipulated for replica management. It can be downloaded from:

   http://www.globus.org/gt2/replica.schema.txt

   Copy this file to $GLOBUS_LOCATION/etc/openldap/schema/replica.schema.

   Edit $GLOBUS_LOCATION/etc/openldap/slapd.conf to reflect your site's requirements (for all bolded entries).

```
# See slapd.conf(5) for details on configuration options.
# This file should NOT be world readable.
#
include /usr/local/globus/etc/openldap/schema/core.schema
include /usr/local/globus/etc/openldap/schema/replica.schema
pidfile /usr/local/globus/var/slapd.pid
argsfile /usr/local/globus/var/slapd.args

########################################################
 # ldbm database definitions
 ########################################################
database ldbm
 suffix "dc=itso-maya,dc=com"
 rootdn "cn=Manager, dc=itso-maya,dc=com "
 rootpw globus
 directory /usr/local/globus/var/openldap-ldbm
 index objectClass eq
```

   Be sure to include the following two lines in the file near the top:

```
schemacheck off
include /usr/local/globus/etc/openldap/schemas/replica.schema
```

2. Start the LDAP daemon:

```
export  LD_LIBRARY_PATH=$GLOBUS_LOCATION/etc
$GLOBUS_LOCATION/libexec/slapd -f $GLOBUS_LOCATION/etc/openldap/slapd.conf
```

3. The LDAP daemon sends a message to the syslogd daemon though the local4 facility. Add the following line in /etc/syslogd.conf:

```
Local4.*              /var/log/ldap.log
```

   Issue `service syslogd reload` to enable LDAP error messages. For any issues regarding the LDAP server, you can check /var/log/ldap.log to determine what the problem might be.

4. Initialize the catalog.

   a. Open a shell and issue:

```
. $GLOBUS_LOCATION/etc/globus-user-env.sh
```

Note: All bold statements are specific to your site and need to be replaced where necessary.

b. Create a file called root.ldif with the following contents:

```
dn: dc=itso-maya, dc=com
objectclass: top
objectclass: GlobusTop
```

c. Create a file called rc.ldif with the following contents:

```
dn: rc=test, dc=itso-maya, dc=com
objectclass: top
objectclass: GlobusReplicaCatalog
objectclass: GlobusTop
rc: test
```

d. Now run the following commands:

```
ldapadd -x -h m0.itso-maya,com  -D   "cn=Manager,dc=itso-maya,dc=com"
-w globus -f root.ldif

ldapadd -x -h m0.itso-maya,com -D  "cn=Manager,dc=itso-maya,dc=com"   -w
globus -f rc.ldif

ldapsearch -h ldap.server.com -b "dc=itso-maya.com" objectclass=*
```

You should see the following in the output:

```
dn: dc=itso-maya,dc=com
objectclass: top
objectclass: GlobusTop

dn: rc=test, dc=itso-maya,dc=com
objectclass: top
objectclass: GlobusReplicaCatalog
objectclass: GlobusTop
```

7.6 Summary

The Globus Toolkit 2.2 does not provide a complete data grid solution, but provides all of the components of the infrastructure to efficiently build a secure and robust data grid solution. Major data grid projects are based on or use the Globus Toolkit and have developed data grid solutions suited to their needs.

The Globus Toolkit 2 provides two kinds of services regarding data grid needs:

- For data transfer and access:
  - GASS, which is a simple, multi-protocol file transfer. It is tightly integrated with GRAM
  - GridFTP, which is a protocol and client-server software that provides high-performance and reliable data transfer
- For data replication and management:
  - Replica Catalog, which provides a catalog service for keeping track of replicated data sets
  - Replica Management, which provides services for creating and managing replicated data sets

All these services should not be considered as a complete and integrated Data Grid solution, but they provide APIs and components to application developers to build a data grid solution that will fit their expectation and that will integrate easily with their application.

8

Developing a portal

We have mentioned multiple times that the likely user interface to grid applications will be through portals, specifically Web portals. This chapter shows what such a portal might look like and provides the sample code required to create it.

The assumption is that the grid-specific logic, such as job brokers, job submission, and so on, has already been written and is available as Java or C/C++ programs that can simply be called from this portal. A few examples of integrating Globus calls with the grid portal are shown.

© Copyright IBM Corp. 2003. All rights reserved.

8.1 Building a simple portal

The simple grid portal has a login screen as shown in Figure 8-1.

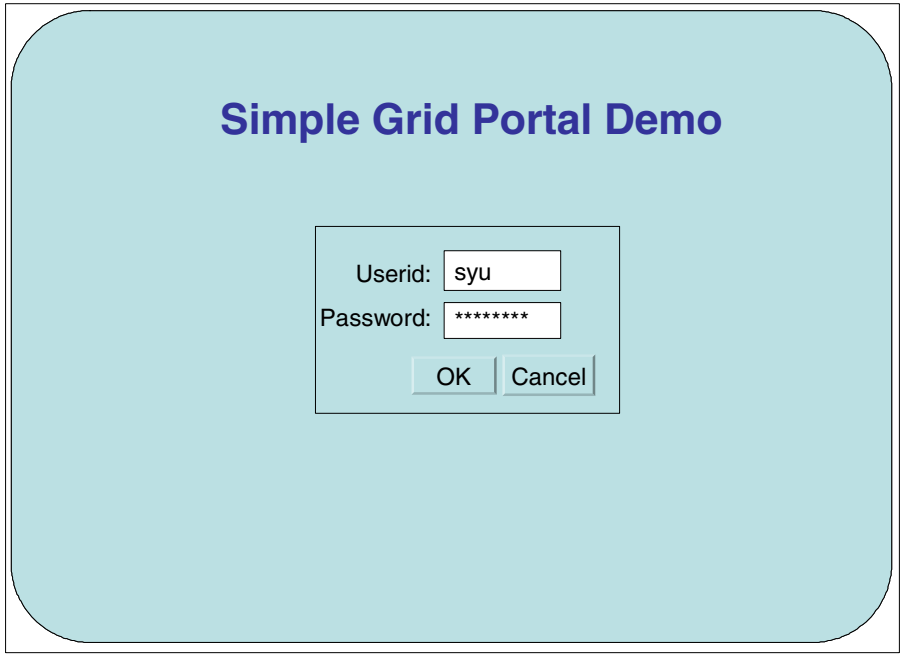


Figure 8-1 Sample grid portal login screen

After the user has successfully authenticated with a user ID and password, the welcome screen is presented, as shown in Figure 8-2 on page 217.

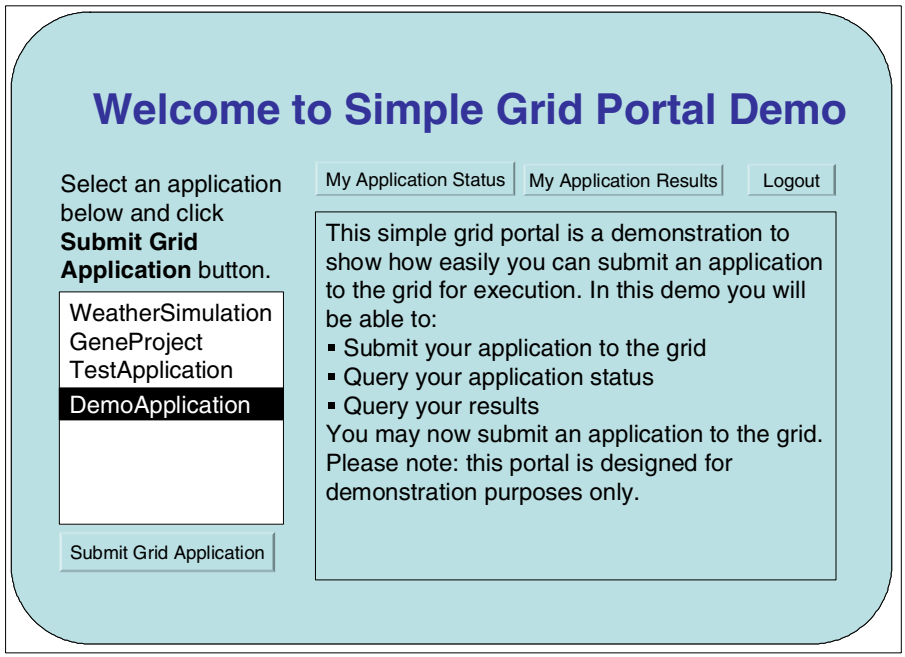

Welcome to Simple Grid Portal Demo
Select an application below and click **Submit Grid Application** button.
WeatherSimulation
GeneProject
TestApplication
DemoApplication
Submit Grid Application
My Application Status
My Application Results
Logout
This simple grid portal is a demonstration to show how easily you can submit an application to the grid for execution. In this demo you will be able to:
▪ Submit your application to the grid
▪ Query your application status
▪ Query your results
You may now submit an application to the grid.
Please note: this portal is designed for demonstration purposes only.

Figure 8-2 Simple grid portal welcome screen

From the left portion of the welcome screen, the user is able to submit an application by selecting a grid application from the list and clicking **Submit Grid Application**. With the buttons on the top right portion of the screen, the user is able to retrieve information about the grid application such as the status and the run results. Clicking the Logout button shows the login screen again.

Let us see how this may be implemented by using an application server such as WebSphere Application Server. Figure 8-3 on page 218 shows the high-level view of a simple grid portal application flow.

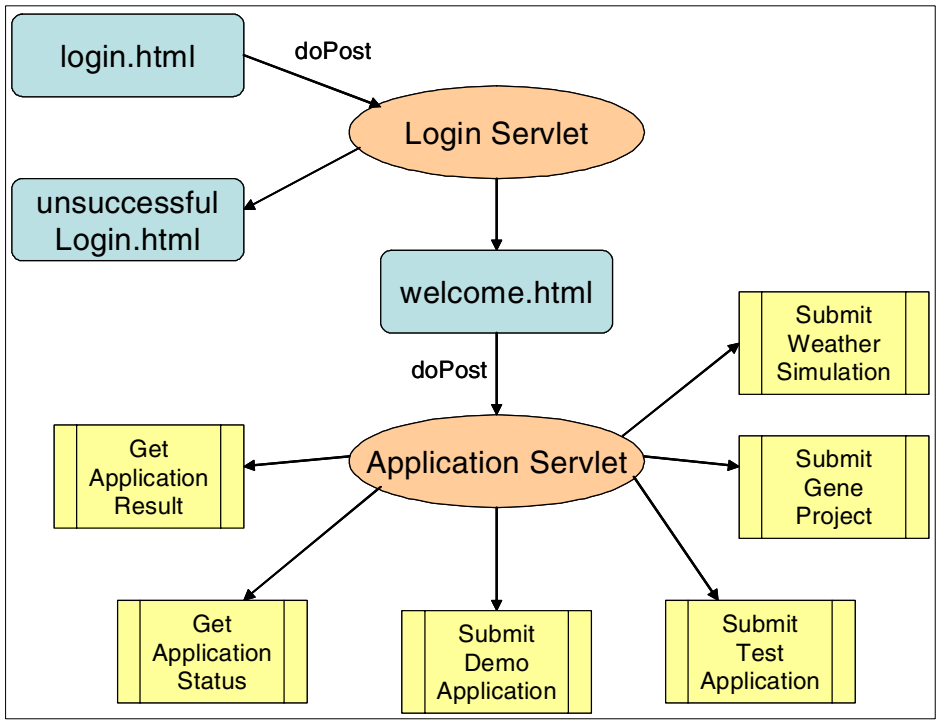


Figure 8-3 Simple grid portal application flow

The login.html produces the login screen, where the user enters the user ID and password. The control is passed to the Login Servlet with the user ID and password as input arguments. The user is authenticated by the servlet. If successful, the user is presented with a welcome screen with the welcome.html file. Otherwise, the user is presented with an unsuccessful login screen with the unsuccessfulLogin.html file. See Figure 8-4 on page 219.

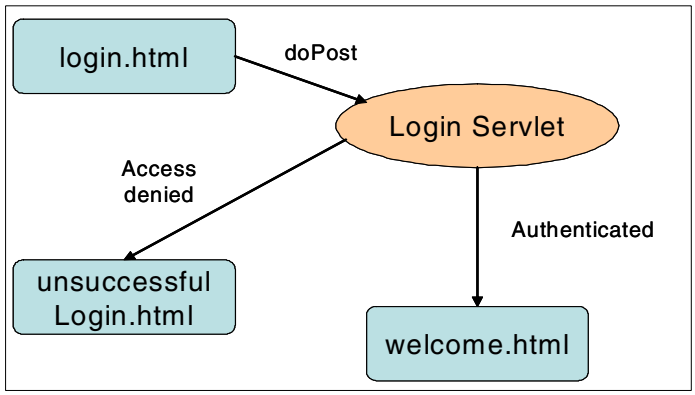


Figure 8-4 Simple grid portal login flow

Example 8-1 shows sample script code for the login.html to display the sample login screen.

Tip: The login servlet is associated with login.html with the following statement:

```
<FORM name="form" method="post" action="Login">
```

Example 8-1 Sample login.html script

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
   type="text/css">
<TITLE>login.html</TITLE>
</HEAD>
<BODY>
<FORM name="form" method="post" action="Login">
<TABLE border="1" width="662" height="296">
   <TBODY>
      <TR>
         <TD width="136" height="68"></TD>
         <TD width="518" height="68"></TD>
      </TR>
      <TR>
         <TD width="136" height="224"></TD>
         <TD width="518" height="224">
```

```
         <P>Userid: <INPUT type="text" name="userid" size="20"
maxlength="20"></P>
         <P>Password: <INPUT type="password" name="password" size="20"
            maxlength="20"></P>
         <INPUT type="submit" name="loginOkay" value="Login"></TD>
      </TR>
   </TBODY>
</TABLE>
</FORM>
</BODY>
</HTML
```

Example 8-2 shows sample Login.java servlet code.

Tip: The class definition clause `extends HttpServlet` distinguishes a servlet. Another distinguishing mark of a servlet is the input parameters (HttpServletRequest req, HttpServletResponse res).

The arguments from the login.html are passed to the Login.java servlet through the HttpServletRequest req parameter. When the authentication is successful, the control is passed to wecome.html using a redirect command, rd.forward(request, response).

Example 8-2 Sample Login.java servlet code

```
package com.ibm.itso.mygridportal.web;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * @version 1.0
 * @author
 */
public class Login extends HttpServlet {

   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
```

```
   }

   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
   }
   public void performTask(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException {

      /**
      * Add your authentication code here
      */
      /**
      * If authentication successful
      */
      System.out.println("Login: forwarding to welcome page");
      try {
         RequestDispatcher rd =
            getServletContext().getRequestDispatcher("welcome.html");
         rd.forward(request, response);
      } catch (java.io.IOException e) {
         System.out.println(e);
      }
      /**
      * If authentication failed
      */
      try {
         RequestDispatcher rd =
            getServletContext().getRequestDispatcher \
               ("unsuccessfulLogin.html");
         rd.forward(request, response);
      } catch (java.io.IOException e) {
         System.out.println(e);
      }
   }
}
```

The file welcome.html produces the welcome screen. From here, the user may select a grid application from the list and submit. Clicking **Submit Grid Application** button sends control to the application servlet. The selected grid application is identified in the servlet and appropriate routines are invoked as shown in Figure 8-5 on page 222.

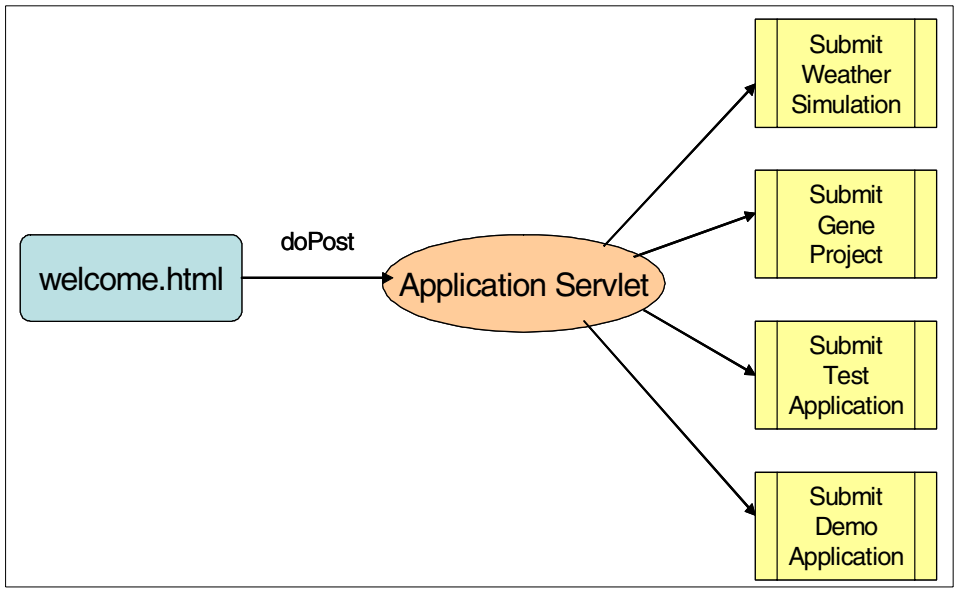


Figure 8-5 Simple grid portal application submit flow

The welcome.html script is provided in Example 8-3 on page 223.

Tip: The application servlet is associated with welcome.html with following statement:

```
<FORM name="form" method="post" action="Application">
```

Tip: The application selection list is produced by the nested statements:

```
<SELECT size="4" name="appselect">
   <OPTION value="weather">WeatherSimulation</OPTION>
   <OPTION value="gene">GeneProject</OPTION>
   <OPTION value="test">TestApp</OPTION>
   <OPTION value="demo" selected>DemoApp</OPTION>
</SELECT>
```

Example 8-3 Simple grid portal welcome.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
   type="text/css">
<TITLE>welcome.jsp</TITLE>
</HEAD>
<BODY>
<FORM name="form" method="post" action="Application">
<H1 align="center">Welcome to Grid Portal Demo</H1>
<TABLE border="1" width="718" height="262">
   <TBODY>
      <TR>
         <TD width="209" height="37"></TD>
         <TD width="501" height="37">
         <TABLE border="1" width="474">
            <TBODY>
               <TR>
                  <TD width="20%"><INPUT type="submit" name="status"
                     value="My Application Status"></TD>
                  <TD width="20%"><INPUT type="submit" name="result"
                     value="My Application Results"></TD>
                  <TD width="20%"></TD>
                  <TD width="20%"></TD>
                  <TD width="20%"><INPUT type="submit" name="logout"
                     value="Logout"></TD>
               </TR>
            </TBODY>
         </TABLE>
         </TD>
      </TR>
      <TR>
         <TD width="209" height="225" valign="top">
         <P>Select an  application and click Grid Application Application
         button below.</P><SELECT size="4" name="appselect">
            <OPTION value="weather">WeatherSimulation</OPTION>
            <OPTION value="gene">GeneProject</OPTION>
            <OPTION value="test">TestApp</OPTION>
            <OPTION value="demo" selected>DemoApp</OPTION>
```

```
         </SELECT><br>
         <INPUT type="submit" name="submit" value="Submit Grid
Application"><BR>

         </TD>
         <TD width="501" height="225">
         <P>This grid portal is a demontration to show how easily you can
submit an application to the grid for execution. In this demo you will be able
to:
         </P>
         <UL>
            <LI>Submit your application to the grid</LI>
            <LI>Query your application status</LI>
            <LI>Query your results</LI>
         </UL>

         <P>You may now submit an application to the grid. <BR>Please note:
this portal is designed for demonstration purposes only.
         </P>
         </TD>
      </TR>
   </TBODY>
</TABLE>
</FORM>
</BODY>
</HTML>
```

The Application.java servlet code is shown in Example 8-4 on page 225.

Tip: Determine which application was selected:

```
private void submitApplication() {
   if (appselect[0].equals("weather"))
      submitWeather();
   else if (appselect[0].equals("gene"))
      submitGene();
   else if (appselect[0].equals("test"))
      submitTest();
   else if (appselect[0].equals("demo"))
      submitDemo();
   else
      invalidSelection();
}
```

Tip: Determine if the **Submit Grid Application** button was checked:

```
   String[] submit;
   String[] appselect;
      try {
         // Which button selected?
         submit = req.getParameterValues("submit");
         // Which application was selected?
         appselect = req.getParameterValues("appselect");

         if (submit != null && submit.length > 0)
            submitApplication(); // submit Application.
            ...
         else
            invalidInput();
      } catch (Throwable theException) {
         // uncomment the following line when unexpected exceptions are
occuring to aid in debugging the problem
         // theException.printStackTrace();
         throw new ServletException(theException);
      }
```

Example 8-4 Simple grid portal Application.java servlet code

```
// 5630-A23, 5630-A22, (C) Copyright IBM Corporation, 2003
// All rights reserved. Licensed Materials Property of IBM
// Note to US Government users: Documentation related to restricted rights
// Use, duplication or disclosure is subject to restrictions set forth in GSA
ADP Schedule with IBM Corp.
// This page may contain other proprietary notices and copyright information,
the terms of which must be observed and followed.
//
// This program may be used, executed, copied, modified and distributed
// without royalty for the purpose of developing, using,
// marketing, or distributing.
//
package com.ibm.itso.mygridportal.web;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.util.*;

/**
```

```
 * @version 1.0
 * @author
 */
public class Application extends HttpServlet {
   HttpServletRequest req; //request
   HttpServletResponse res; //response
   JSPBean jspbean = new JSPBean();
   PrintWriter out;
   String[] submit;
   String[] getresult;
   String[] getstatus;
   String[] logout;
   String[] appselect;
   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
   }

   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
   }

   public void performTask(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException {

      req = request;
      res = response;

      res.setContentType("text/html");
      res.setHeader("Pragma", "no-cache");
      res.setHeader("Cache-control", "no-cache");
      try {
         out = res.getWriter();
      } catch (IOException e) {
         System.err.println("Application:getWriter:" + e);
      }

      // --- Read and validate user input, initialize. ---
```

```
      try {
         // Which button selected?
         submit = req.getParameterValues("submit");
         getresult = req.getParameterValues("result");
         getstatus = req.getParameterValues("status");
         logout = req.getParameterValues("logout");

         // Which application was selected?
         appselect = req.getParameterValues("appselect");

         if (submit != null && submit.length > 0)
            submitApplication(); // submit Application.
         else if (getresult != null && getresult.length > 0)
            getResult(); // get run result.
         else if (getstatus != null && getstatus.length > 0)
            getStatus(); // get Application status.
         else if (logout != null && logout.length > 0)
            doLogout(); // logout.
         else
            invalidInput();
      } catch (Throwable theException) {
         // uncomment the following line when unexpected exceptions are
occuring to aid in debugging the problem
         // theException.printStackTrace();
         throw new ServletException(theException);
      }

   }

   private void submitApplication() {
      if (appselect[0].equals("weather"))
         submitWeather();
      else if (appselect[0].equals("gene"))
         submitGene();
      else if (appselect[0].equals("test"))
         submitTest();
      else if (appselect[0].equals("demo"))
         submitDemo();
      else
         invalidSelection();
   }

   private void submitWeather() {
   /**
   * Add code to submit the weather application here
   */
   }

   private void submitGene() {
```

```
/**
* Add code to submit the gene application here
*/
}

private void submitTest() {
/**
* Add code to submit the test application here
*/
}

private void submitDemo() {
/**
* Add code to submit the demo application here
*/
}

private void getResult() {
/**
* Add code to get the Application results here
*/
}

private void getStatus() {
/**
* Add code to get the Application status here
*/
}

private void doLogout() {
   System.out.println("doLogout: forwarding to login page");
   try {
      RequestDispatcher rd =
         getServletContext().getRequestDispatcher("login.html");
      rd.forward(req, res);
   } catch (javax.servlet.ServletException e) {
      System.out.println(e);
   } catch (java.io.IOException e) {
      System.out.println(e);
   }
}

private void invalidSelection() {
   // Something was wrong with the client input
   try {
      RequestDispatcher rd =
         getServletContext().getRequestDispatcher("invalidSelection.html");
      rd.forward(req, res);
   } catch (javax.servlet.ServletException e) {
```

```
            System.out.println(e);
        } catch (java.io.IOException e) {
            System.out.println(e);
        }
    }

    private void invalidInput() {
        // Something was wrong with the client input
        try {
            RequestDispatcher rd =
                getServletContext().getRequestDispatcher("invalidInput.html");
            rd.forward(req, res);
        } catch (javax.servlet.ServletException e) {
            System.out.println(e);
        } catch (java.io.IOException e) {
            System.out.println(e);
        }
    }

    private void sendResult(String[] list) {
        int size = list.length;
        for (int i = 0; i < size; i++) {
            String s = list[i];
            out.println(s + "<br>");
            //System.out.println("s=" + s); //trace
        } //end for
    }

}
```

From the welcome screen, the user may also request application status, application results, and logout. Figure 8-6 on page 230 shows the flow. When the user clicks **My Application Status**, **My Application Results**, or **Logout**, the Application Servlet is called.

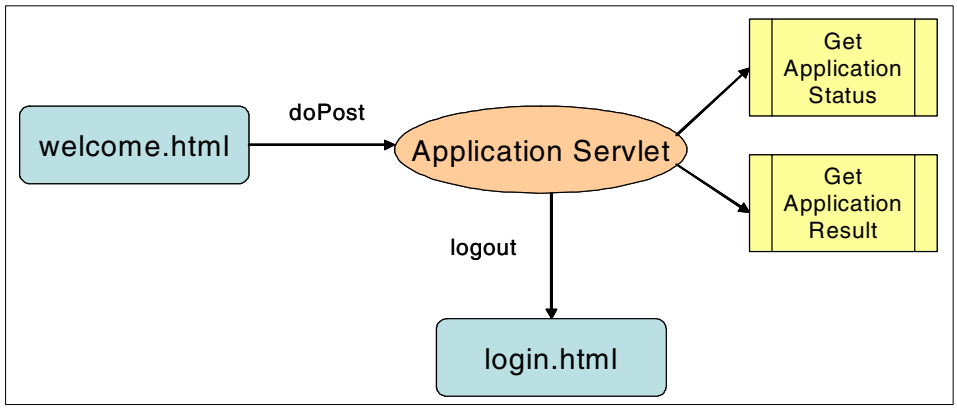


Figure 8-6 Simple grid portal application information and logout flow

Tip: Determine which button is pressed from welcome.html:

```
<TD width="20%"><INPUT type="submit" name="status"
   value="My Application Status"></TD>
<TD width="20%"><INPUT type="submit" name="data"
   value="My Application Results"></TD>
<TD width="20%"></TD>
<TD width="20%"></TD>
<TD width="20%"><INPUT type="submit" name="logout"
   value="Logout"></TD>
```

Tip: Determine which button is pressed:

```
   String[] getresult;
   String[] getstatus;
   String[] logout;
      try {
         // Which button selected?
         getresult = req.getParameterValues("result");
         getstatus = req.getParameterValues("status");
         logout = req.getParameterValues("logout");

         if (submit != null && submit.length > 0)
            submitApplication(); // submit Application.
         else if (getresult != null && getresult.length > 0)
            getResult(); // get application run results.
         else if (getstatus != null && getstatus.length > 0)
            getStatus(); // get application status.
         else if (logout != null && logout.length > 0)
            doLogout(); // logout.
         else
            invalidInput();
      } catch (Throwable theException) {
         // uncomment the following line when unexpected exceptions are
occuring to aid in debugging the problem
         // theException.printStackTrace();
         throw new ServletException(theException);
      }
```

Tip: How to redirect to an html page:

```
private void doLogout() {
   try {
      RequestDispatcher rd =
         getServletContext().getRequestDispatcher("login.html");
      rd.forward(req, res);
   } catch (javax.servlet.ServletException e) {
      System.out.println(e);
   } catch (java.io.IOException e) {
      System.out.println(e);
   }
}
```

8.2 Integrating portal function with a grid application

This section describes some techniques for integrating the portal with a grid-enabled application.

8.2.1 Add methods to execute the Globus commands

The simplest and most obvious integration is to be able to launch or execute Globus commands via the Web interface. From a Java servlet, you may need to launch Globus commands written in C. Let us see how this is accomplished.

Running non-Java commands from Java code

Our portal code is written in Java. If the grid commands and the Globus commands are also Java classes then this section can be skipped. There are two possible ways to do this. One way is to use the Java native method, which can be difficult to implement. The second way is to use the exec() method of the Runtime class. This method is easier to implement and our choice used in the sample code. Either option will lose platform independence.

The sample code in Example 8-5 shows how to execute the grid commands and Globus commands from the Web portal Java code.

Tip: The method of running non-Java commands from Java code is:

```
Proc p = Runtime.getRuntime().exec(cmd);
```

Tip: Below is the method of properly passing parameter inputs or sub-commands to the non-Java command. The exec() method has trouble handling a single string passed as one command. With the string array, “/bin/su” , “-c”, and “subCmd” are treated separately and execute correctly.

```
String[] cmd = {"/bin/su", "-c", subCmd, "-", "mOuser"};
cmdResult = doRun(cmd);
```

Example 8-5 Sample code to run non-Java commands from Web portal

```
public String[] doRun(String[] cmd) throws IOException {
   ArrayList cmdOutput;
   Process p;
   InputStream cmdOut;
   BufferedReader brOut;
   InputStream cmdErr;
   BufferedReader brErr;
   String line;
```

```
   cmdOutput = new ArrayList();

   p = Runtime.getRuntime().exec(cmd);

   cmdOut = p.getInputStream();
   brOut = new BufferedReader(new InputStreamReader(cmdOut));

   // get the command output
   cmdOutput.clear();
   while ((line = brOut.readLine()) != null)
      cmdOutput.add(line);

   cmdErr = p.getErrorStream();
   brErr = new BufferedReader(new InputStreamReader(cmdErr));

   // get error message
   while ((line = brErr.readLine()) != null)
      cmdOutput.add(line);

   try {
      p.waitFor();
   } catch (InterruptedException e) {
      System.out.println("Command "+cmd+" interrupted." + e);
   }
   return (String[]) cmdOutput.toArray(new String[0]);
}
```

Security

In the sample code in Example 8-6, the **grid-proxy-init** command is executed to get a valid proxy. The **"su -c command - userid"** is used to switch to the correct user to run the command.

Example 8-6 Sample code to run grid-proxy-init from Web portal

```
public String[] doProxy() {
   String[] cmdResult = null;
   String subCmd = "echo mOuser | grid-proxy-init -pwstdin";
   try {
      String[] cmd = {"/bin/su", "-c", subCmd, "-", "mOuser"};
      cmdResult = doRun(cmd);
   }
   catch (IOException e) {
      System.out.println("doProxy: "+e);
   }
return cmdResult;
}
```

Submit grid application for execution

Example 8-7 shows sample code to submit a job to the grid for execution. The "mdsHost" is a fully qualified host name where the application will be submitted. A valid grid host may be found from the output of the **grid-info-search** command. The "-stage" parameter is required if the application resides on the submission machine and not on the execution machine. The jobId is returned after successful submission, as shown in the tip below.

Tip: The job ID is returned by the **globus-job-submit** command, as shown below. This job ID will be used to get the status and run the result, as shown in the next section.

```
https://a1.itso-apache.com:41418/2772/1049325984/
```

Example 8-7 Sample code to submit a job from Web portal

```
public String doSubmit(String mdsHost) {
   String[] cmdResult = null;
   String subCmd = "globus-job-submit "+mdsHost+" -stage
/home/mOuser/test.sh";
   try {
      String[] cmd = {"/bin/su", "-c", subCmd, "-", "mOuser"};
      cmdResult = doRun(cmd);
   }
   catch (IOException e) {
      System.out.println("doSubmit: "+e);
   }
   return cmdResult[0];
}
```

Get application status

The sample code in Example 8-8 shows how to retrieve the job status. The job ID obtained from the job submission is input to the doGetStatus() method. When the job is running, the job status `ACTIVE` is returned. When the job completes, the job status `DONE` is returned.

Example 8-8 Sample code to get job status from Web portal

```
public String doGetStatus(String jobId) {
   String[] cmdResult = null;
   String subCmd = "globus-job-status "+jobId;
   try {
      String[] cmd = {"/bin/su", "-c", subCmd, "-", "mOuser"};
      cmdResult = doRun(cmd);
   }
   catch (IOException e) {
      System.out.println("doGetStatus: "+e);
```

```
    }
    return cmdResult[0];
}
```

Get application run results

The sample code to get the run result is shown in Example 8-9. The job ID obtained at job submission is used as input. The run results are cached after the job completes and returns.

Example 8-9 Sample code to get job output from Web portal

```
public String[] doGetResult(String jobId) {
    String[] cmdResult = null;
    String subCmd = "globus-job-get-output "+jobId;
    try {
        String[] cmd2 = {"/bin/su", "-c", subCmd, "-", "mOuser"};
        cmdResult = doRun(cmd2);
    }
    catch (IOException e) {
        System.out.println("doGetResult: "+e);
    }
return cmdResult;
}
```

Cancel or clean a job

Use **`globus-job-cancel jobId`** to kill a running job. The job ID is obtained when the job is submitted. The cached output from the job is not removed.

Use **`globus-job-clean jobId`** to kill a job if it is running and to remove the cached output on the execution machine.

Tip: It is a good idea to periodically clean up the cache. Globus does not clean up the cached output files. It will be a good idea to set up a routine administrative procedure to periodically clean up the cache with the command **`rm globus_gass_cache_*`** in the .globus/.gass_cache directory on the execution machine.

Repository of job ids for submitted applications

Example 8-10 on page 236 shows a class that could be used as a container for job IDs for all submitted applications.

Example 8-10 Container for submitted job IDs

```
import java.util.*;

public class JSPBean {

   private ArrayList jobIds;

   public JSPBean() {
      jobIds = new ArrayList();
   }

   public void addJobId(String id) {
      jobIds.add(id);
   }

   public String[] getJobIds() {
      return (String[])jobIds.toArray(new String[0]);
   }
}
```

8.2.2 Putting it together

Example 8-11 includes the complete source code for the portal as has been described throughout this chapter.

Example 8-11 Complete source code for grid portal sample

```
package com.ibm.itso.mygridportal.web;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.*;
import java.util.*;

/**
 * @version 1.0
 * @author
 */
public class Application extends HttpServlet {

   HttpServletRequest req; //request
   HttpServletResponse res; //response
```

```
   String mdsHost = "a1.itso-apache.com";
   String testApp = "/home/mOuser/test.sh";
   JSPBean jspBean = new JSPBean();
   PrintWriter out;
   String[] run;
   String[] getresult;
   String[] getstatus;
   String[] logout;
   String[] appselect;
   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
   }

   /**
   * @see javax.servlet.http.HttpServlet#void
(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
   */
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
      performTask(req, resp);
   }

   public void performTask(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException {

      req = request;
      res = response;

      res.setContentType("text/html");
      res.setHeader("Pragma", "no-cache");
      res.setHeader("Cache-control", "no-cache");
      try {
         out = res.getWriter();
      } catch (IOException e) {
         System.err.println("Job:getWriter:" + e);
      }

      // --- Read and validate user input, initialize. ---
      try {
         // Which button selected?
         run = req.getParameterValues("run");
         getresult = req.getParameterValues("data");
```

```
            getstatus = req.getParameterValues("status");
            logout = req.getParameterValues("logout");

            // Which application was selected?
            appselect = req.getParameterValues("appselect");

            if (run != null && run.length > 0)
               submitApplication(); // submit job.
            else if (getresult != null && getresult.length > 0)
               getResult(); // get run result.
            else if (getstatus != null && getstatus.length > 0)
               getStatus(); // get job status.
            else if (logout != null && logout.length > 0)
               doLogout(); // logout.
            else
               invalidInput();
         } catch (Throwable theException) {
            // uncomment the following line when unexpected exceptions are
occuring to aid in debugging the problem
            // theException.printStackTrace();
            throw new ServletException(theException);
         }

      }

      private void submitApplication() {
         if (appselect[0].equals("weather"))
            submitWeather();
         else if (appselect[0].equals("gene"))
            submitGene();
         else if (appselect[0].equals("test"))
            submitTest();
         else if (appselect[0].equals("demo"))
            submitDemo();
         else
            invalidSelection();
      }

      private void submitWeather() {
         String title = "Submit weather application";
         out.println("<html>");
         out.println("<head>");
         out.println("<title>" + title + "</title>");
         out.println("</head>");
         out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
         out.println("<h1 align=\"center\">" + title + "</h1>");
         out.println("</body>");
         out.println("</html>");
      }
```

```
private void submitGene() {
   String title = "Submit gene application";
   out.println("<html>");
   out.println("<head>");
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
   out.println("<h1 align=\"center\">" + title + "</h1>");
   out.println("</body>");
   out.println("</html>");
}

private void submitTest() {
   String title = "Submit test application";
   out.println("<html>");
   out.println("<head>");
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
   out.println("<h1 align=\"center\">" + title + "</h1>");
   sendResult(doProxy());
   jspBean.addJobId(doSubmit(mdsHost, testApp));
   out.println("</body>");
   out.println("</html>");
}

private void submitDemo() {
   String title = "Submit Demo application";
   out.println("<html>");
   out.println("<head>");
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
   out.println("<h1 align=\"center\">" + title + "</h1>");
   out.println("</body>");
   out.println("</html>");
}

private void getStatus() {
   String[] myJobIds;
   String title = "Grid Job Status";
   out.println("<html>");
   out.println("<head>");
   out.println("<title>" + title + "</title>");
   out.println("</head>");
   out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
   out.println("<h1 align=\"center\">" + title + "</h1>");
   myJobIds = jspBean.getJobIds();
```

```
        int size = myJobIds.length;
        if (size > 0) {
           out.println("<table border=1>");
             out.println("<tr>");
             out.print("<td><b>Job Id</td>");
             out.println("<td>Job Status</b></td>");
             out.println("</tr>");
           for (int i = 0; i < size; i++) {
              String myJobId = myJobIds[i];
              out.println("<tr>");
              out.print("<td>" + myJobId + "</td>");
              out.println("<td>" + doGetStatus(myJobId) + "</td>");
              out.println("</tr>");
           }
           out.println("</table>");
        }

        out.println("</body>");
        out.println("</html>");
     }

     private void getResult() {
        String[] myJobIds;
        String title = "Grid Application Run Result";
        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
        out.println("<h1 align=\"center\">" + title + "</h1>");
        myJobIds = jspBean.getJobIds();

        int size = myJobIds.length;
        if (size > 0) {
           out.println("<table border=1>");
             out.println("<tr>");
             out.print("<td><b>Job Id</td>");
             out.println("<td>Job Results</b></td>");
             out.println("</tr>");
           for (int i = 0; i < size; i++) {
              String myJobId = myJobIds[i];
              out.println("<tr>");
              out.print("<td>" + myJobId + "</td>");
              out.println("<td>");
              sendResult(doGetResult(myJobId));
              out.println("</td>");
              out.println("</tr>");
           }
```

```
            out.println("</table>");
        }

        out.println("</body>");
        out.println("</html>");
    }

    private void doLogout() {
        System.out.println("doLogout: forwarding to login page");
        try {
            RequestDispatcher rd =
                getServletContext().getRequestDispatcher("login.html");
            rd.forward(req, res);
        } catch (javax.servlet.ServletException e) {
            System.out.println(e);
        } catch (java.io.IOException e) {
            System.out.println(e);
        }
    }

    private void invalidSelection() {
        // Something was wrong with the client input
        System.out.println("invalid selection");
        String title = "Invalid selection";
        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
        out.println("<h1 align=\"center\">" + title + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }

    private void invalidInput() {
        // Something was wrong with the client input
        System.out.println("invalid input");
        String title = "Invalid Input";

        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + title + "</title>");
        out.println("</head>");
        out.println("<body bgcolor=\"#cfd09d\" text=\"#000099\">");
        out.println("<h1 align=\"center\">" + title + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
    private void sendResult(String[] list) {
```

```
    int size = list.length;
    for (int i = 0; i < size; i++) {
       String s = list[i];
       out.println(s + "<br>");
       //System.out.println("s=" + s); //trace
    } //end for
}
public String[] doProxy() {
    String[] cmdResult = null;
    String subCmd = "echo mOuser | grid-proxy-init -pwstdin";
    try {
       String[] cmd = { "/bin/su", "-c", subCmd, "-", "mOuser" };
       cmdResult = doRun(cmd);
    } catch (IOException e) {
       System.out.println("doProxy: " + e);
    }
    return cmdResult;
}
public String doSubmit(String mdsHost, String appName) {
    String[] cmdResult = null;
    String subCmd =
       "globus-job-submit " + mdsHost + " -stage " + appName;
    try {
       String[] cmd = { "/bin/su", "-c", subCmd, "-", "mOuser" };
       cmdResult = doRun(cmd);
    } catch (IOException e) {
       System.out.println("doSubmit: " + e);
    }
    return cmdResult[0];
}
public String doGetStatus(String jobId) {
    String[] cmdResult = null;
    String subCmd = "globus-job-status " + jobId;
    try {
       String[] cmd = { "/bin/su", "-c", subCmd, "-", "mOuser" };
       cmdResult = doRun(cmd);
    } catch (IOException e) {
       System.out.println("doGetStatus: " + e);
    }
    return cmdResult[0];
}
public String[] doGetResult(String jobId) {
    String[] cmdResult = null;
    String subCmd = "globus-job-get-output " + jobId;
    try {
       String[] cmd2 = { "/bin/su", "-c", subCmd, "-", "mOuser" };
       cmdResult = doRun(cmd2);
    } catch (IOException e) {
       System.out.println("doGetResult: " + e);
```

```
        }
        return cmdResult;
    }
    public String[] doRun(String[] cmd) throws IOException {
        ArrayList cmdOutput;
        Process p;
        InputStream cmdOut;
        BufferedReader brOut;
        InputStream cmdErr;
        BufferedReader brErr;
        String line;

        cmdOutput = new ArrayList();

        p = Runtime.getRuntime().exec(cmd);

        cmdOut = p.getInputStream();
        brOut = new BufferedReader(new InputStreamReader(cmdOut));

        // get the command output
        cmdOutput.clear();
        while ((line = brOut.readLine()) != null)
            cmdOutput.add(line);

        cmdErr = p.getErrorStream();
        brErr = new BufferedReader(new InputStreamReader(cmdErr));

        // get error message
        while ((line = brErr.readLine()) != null)
            cmdOutput.add(line);

        try {
            p.waitFor();
        } catch (InterruptedException e) {
            System.out.println("Command " + cmd + " interrupted." + e);
        }

        return (String[]) cmdOutput.toArray(new String[0]);

    }
}
```

8.3 Summary

This chapter has provided examples and tips for creating a Web portal that cold be used as an interface for a grid environment. It is meant as a sample on which readers can build a more robust implementation that meet their specific needs.

9

Application examples

This chapter provides several examples of simple applications that have been enabled to run in a grid environment. These samples provide many examples of techniques and help solidify concepts that may be useful to the application developer. They are provided as programming examples that may be useful to readers in developing more sophisticated applications for their businesses.

© Copyright IBM Corp. 2003. All rights reserved.

9.1 Lottery simulation program

The first application simply utilizes the GSI-OpenSSH module. Many people will not consider it a true grid application, as it does not use most of the facilities provided by the Globus Toolkit, and their is little ability to manage the application and control its environment. However, it is an example of the power of using the GSI and GSI-OpenSSH packages to allow multiple systems to work together to provide a solution.

After presenting the GSI-OpenSSH version of the program we then provide a true grid-enabled version that does utilize the Globus Toolkit facilities.

9.1.1 Simulate a lottery using gsissh in a shell script

GSI-OpenSSH is a modified version of the OpenSSH client and server that adds support for GSI authentication. GSIssh can of course be used to remotely create a shell on a remote system to run shell scripts or to interactively issue shell commands but it also permits the transfer of files between systems without being prompted for a password and a user ID.

GSIssh installs the following tools in $GLOBUS_LOCATION/bin:

- **`gsissh`** is used to either securely connect to a remote host or to securely execute a program on a remote host.
- **`gsiscp`** is used to securely copy files or directories onto a remote host.
- **`sftp`** is used to securely copy files or directories onto a remote host. **`sftp`** is not related to gsiftp. It is a different protocol that provides encryption and the same commands as the FTP protocol.

`gsiscp` and **`sftp`** do not perform as well as GridFTP, but they can easily be used in a shell script to transfer files from one location to another.

Note: Be careful to use host names for which certificates have been issued; otherwise, you will get the following kind of error messages if you try to use the IP address instead of the host name:

```
21569: gss_init_context failed
(/O=Grid/O=Globus/CN=host/g3.itso-guarani.com) in the context, and the
target name (/CN=host/192.168.0.203)
```

The purpose of this example is to simulate a large number of draws of a lottery and to check if a winning combination was drawn. Each draw consists of eight numbers between one and 60 inclusive.

The program GenerateDraws whose source code GenerateDraws.C is available in “Lottery example” on page 349 is used to generate random draws. It takes one argument, the number of draws that it needs to simulate (1000000, for example). This program is executed on n nodes by using the GSIssh tools.

```
[globus@m0 other]$ ./GenerateDraws 10
3 29 33 39 41 46 54 55
3 6 7 10 16 22 35 50
4 13 20 26 36 47 48 49
8 12 17 22 26 36 50 57
6 20 25 28 32 40 51 55
1 2 9 17 24 45 49 60
12 15 19 20 39 50 59 60
1 4 8 13 29 49 52 57
3 7 14 29 36 43 52 58
6 8 15 18 20 22 27 45
```

GenerateDraws also creates a file named Monitor on the execution node. This file is copied to the submission nodes every five seconds. Monitor contains the percentage of random draws completed and permits the monitoring of the whole application.

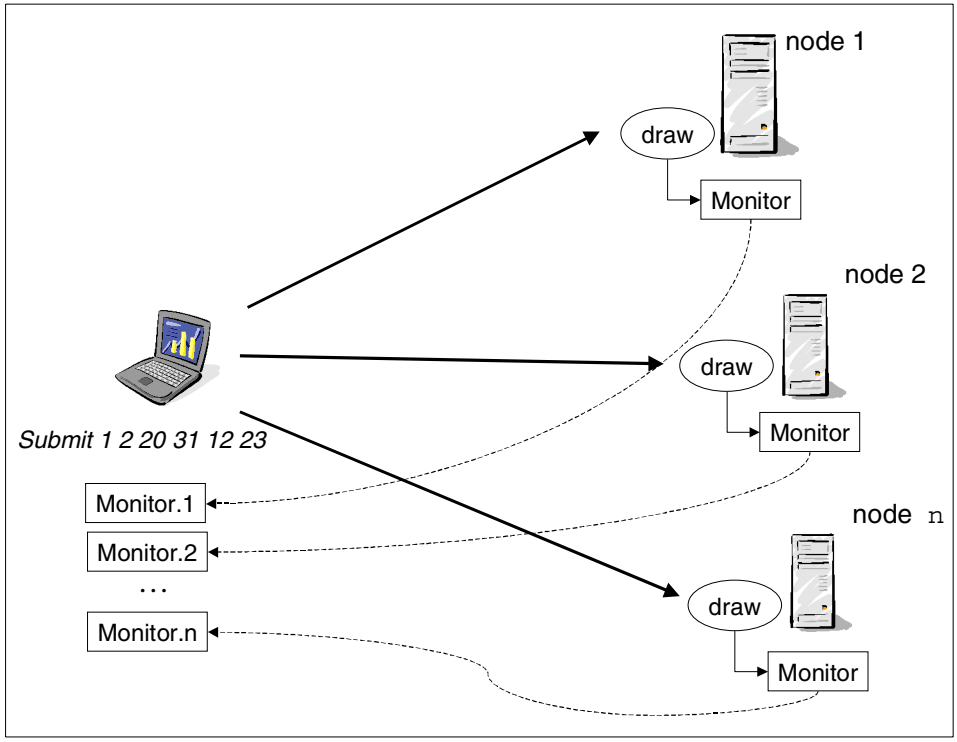


Figure 9-1 Lottery example

The Submit script is used to submit the jobs to the grid. We use the **grep** command to check the output of the GenerateDraws program and to detect if the draw we played is a winner.

Example 9-1 Submit script

```
#the script takes the tested draw as a parameter
#example: ./Submit 3 4 5 32 34 43
n=100000
NodesNumber=10

#temporary working directory on the execution nodes
TMP=.$HOSTNAME


i=0
#the loop variable is used is all the "for" loops
#the format is 1 2 3 4 .... n
loop=""
# use here the broker developped for the publication
# see chapter 8 (mds executable)
for node in $(mds $NodesNumber | xargs)
do
   Nodes[$i]=$node
   loop=${loop}" "${i}
   i=$(( $i + 1 ))
done


echo The number of draws tested is $n
a=$*
#sort the numbers in the specified draw
# 2 45 23 12 32 43 becomes 2 12 23 32 43 45 so that we could use
# grep to test this draw and the ouput of the draw programs.
param=$(echo $a | tr " " "\n" | sort -n | xargs )

# parrarell transfer of the draw executable
# we submit jobs in the background, get their process id
# and uses the wait command to wait for their completion
# this method is also used for the jobs submission
echo Transferring executable files
for i in $loop
do
   gsissh -p 24 ${Nodes[$i]} "[ -d $TMP ] || mkdir $TMP" &
   ProcessID[$i]=$!
done
for i in $loop
do
   wait ${ProcessID[$i]}
```

```
   gsiscp -P 24 GenerateDraws ${Nodes[$i]}:$TMP  &
   ProcessID[$i]=$!
done
for i in $loop
do
   wait ${ProcessID[$i]}
   gsissh -p 24 ${Nodes[$i]} “chmod +x ./$TMP/GenerateDraws” &
   ProcessID[$i]=$!
done
#file should be made executable
#on all the execution nodes
echo Jobs submission to the grid
for i in $loop
do
   wait ${ProcessID[$i]}
   echo ${Nodes[$i]}
   EXE=”cd $TMP;./GenerateDraws $n | grep “‘””$param’” && echo GOT IT on
$HOSTNAME’
   gsissh -p 24 ${Nodes[$i]} “$EXE” &
   ProcessID[$i]=$!
done

#for monitoring, we copy locally the Monitor files
# created on each compute nodes.  This file content the
# percentage of tested draws.  Each files is suffixes by
# the nodes number.  $statusnum is actually the sum of all
# the percentage (Monitor files) devided by 100.  When it
# equals the number of nodes, that means that we have finished
echo Monitoring
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
   sleep 5  #we poll every 5 seconds
   statussum=0
   for i in $loop
   do
      gsiscp -q -P 24 ${Nodes[$i]}:$TMP/Monitor Monitor.$i
      status=$(cat Monitor.$i)
      statussum=$((  $status + $statussum   ))
      echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
   done
   statussum=$((  $statussum  / 100 ))
done
#cleanup the tmp directory
for i in $loop
do
   wait ${ProcessID[$i]}
   gsissh -p 24 ${Nodes[$i]} “rm -fr ./.$TMP” &
```

```
   ProcessID[$i]=$!
done
```

Submit uses the broker (mds program) described in “Broker example” on page 127 to get the host names that it will submit the jobs to by using gsissh:

```
for n in $(mds $NodesNumber | xargs)
do
   Nodes[$i]=$n
   i=$(( $i + 1 ))
done
```

The number of draws per node is determined by the variable n and the number of jobs by the variable NodesNumber:

```
n=100000
NodesNumber=10
```

Sandboxing

Each execution host uses a sandbox directory in each execution node. This directory is created in the local home directory of the user under which the job is executed. This directory is configured by the $TMP variable set up as .<execution hostname>:

```
TMP=.$HOSTNAME
gsissh ${Nodes[$i]} “[ -d $TMP ] || mkdir $TMP” &
```

All file copies and remote execution use the TMP variable in their relative path name to refer to the remote files. This way, each client machine can submit a job without conflicting with another client:

```
gsissh ${Nodes[$i]} “chmod +x ./$TMP/GenerateDraws” &
ProcessID[$i]=$!
gsiscp GenerateDraws ${Nodes[$i]}:$TMP  &
ProcessID[$i]=$!
```

For more granularity, TMP could also use the process ID of the Submit script:

```
TMP=.$HOSTNAME/$$
```

Shell script callback

As we cannot use a callback mechanism in a shell script. We start each command in the background. Therefore, they become non-blocking operations and all commands can be submitted simultaneously.

```
gsiscp GenerateDraws ${Nodes[$i]}:$TMP &
gsissh ${Nodes[$i]} "chmod +x ./$TMP/GenerateDraws" &
```

The **wait** command is used to wait for their completion and acts like a (simple) callback

```
wait ${ProcessID[$i]}
```

Job submission

GenerateDraws needs to be copied to each execution node. Each execution node must have the GSIssh server up and running. **gsiscp** is used to transfer the files and **gsissh** is used to remotely execute the **chmod +x** command, as shown in Example 9-2.

Example 9-2 Job submission using gsissh

```
for i in $loop
do
   gsiscp GenerateDraws ${Nodes[$i]}:$TMP  &
   ProcessID[$i]=$!
done

for i in $loop
do
   wait ${ProcessID[$i]}
   gsissh -p 24 ${Nodes[$i]} "chmod +x ./$TMP/GenerateDraws" &
   ProcessID[$i]=$!
done
for i in $loop
do
   EXE="cd $TMP;./GenerateDraws $n | grep "'"'$param'" && echo GOT IT on\
$HOSTNAME'
   gsissh -p 24 ${Nodes[$i]} "$EXE" &
   ProcessID[$i]=$!
done
for i in $loop
do
   wait ${ProcessID[$i]}
done
```

Monitoring

The monitoring of each job is managed by the Submit script, which reads the content of each file Monitor created by the GenerateDraws executable on each execution node:

```
# for monitoring, we copy locally the Monitor files
# created on each compute nodes.  This file content the
# percentage of tested draws.  Each files is suffixes by
# the nodes number.  $statusnum is actually the sum of all
# the percentage (Monitor files) devided by 100.  When it
# equals the number of nodes, that means that we have finished
```

```
echo Monitoring
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
   sleep 5  #we poll every 5 seconds
   statussum=0
   for i in $loop
   do
      gsiscp -q -P 24 ${Nodes[$i]}:$TMP/Monitor Monitor.$i
      status=$(cat Monitor.$i)
      statussum=$((  $status + $statussum   ))
      echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
   done
   statussum=$((  $statussum  / 100 ))
done
```

How to run it

To use this program we need a valid proxy.

```
echo password | grid-proxy-init -pwstdin
Your identity: /O=Grid/O=Globus/OU=itso-maya.com/CN=globus
Enter GRID pass phrase for this identity:
Creating proxy ..................................... Done
Your proxy is valid until: Thu Feb 27 06:19:17 2003
```

The option -pwstdin permits us to create a proxy without being prompted for a password. This way, the proxy creation can be integrated in the Submit script if needed.

You also need to compile the GenerateDraws.C program and be sure that the GSIssh server is up and running in all nodes. See “Installation” on page 211 for more information.

Note: When connecting to a host for the first time, ssh needs to retrieve its public host key. To bypass this request, you can add the following option to the configuration file $GLOBUS_LOCATION/etc/ssh/ssh_config.

```
StrictHostKeyChecking no
```

Let us perform the test on six numbers instead of eight to have a better chance to win:

```
[globus@m0 other]$ ./Submit 1 4 10 15 20 34
The number of draws tested is 100000
Transferring executable files
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
```

```
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
GenerateDraws        100% |*******************************| 52775       00:00
Jobs submission to the grid
d2.itso-apache.com
a1.itso-cherokee.com
c2.itso-cherokee.com
c1.itso-cherokee.com
t1.itso-tupi.com
t3.itso-tupi.com
d1.itso-apache.com
a2.itso-apache.com
b2.itso-bororos.com
t2.itso-tupi.com
Monitoring

d2.itso-dakota.com:Monitor 37 %
a1.itso-apache.com:Monitor 46 %
c2.itso-cherokee.com:Monitor 41 %
c1.itso-cherokee.com:Monitor 41 %
t1.itso-tupi.com:Monitor 51 %
t3.itso-tupi.com:Monitor 43 %
d1.itso-dakota.com:Monitor 47 %
a2.itso-apache.com:Monitor 49 %
b2.itso-bororos.com:Monitor 40 %
t2.itso-tupi.com:Monitor 55 %

1 4 10 15 20 34 47 57
GOT IT on a1.itso-apache.com
d2.itso-dakota.com:Monitor 97 %
a1.itso-apache.com:Monitor 100 %
c2.itso-cherokee.com:Monitor 100 %
c1.itso-cherokee.com:Monitor 100 %
t1.itso-tupi.com:Monitor 100 %
t3.itso-tupi.com:Monitor 99 %
d1.itso-dakota.com:Monitor 100 %
a2.itso-apache.com:Monitor 100 %
b2.itso-bororos.com:Monitor 84 %
t2.itso-tupi.com:Monitor 100 %
```

9.1.2 Simulate a lottery using Globus commands

We propose to implement the previous example by using Globus Toolkit 2.2 commands:

- **`globusrun`** will be used to submit the job. The type of the job will be a multi-request query.
- **`globus-url-copy`** will be used to copy the Monitor file from the execution nodes to the submission nodes.
- **`globus-gass-server`** will start a GASS server on the execution nodes that will be used to copy the Monitor file generated by **`GenerateDrawsGlobus`** and used to monitor the status of the job.

All the programs used in the previous example are slightly modified but are still used for the same purpose. The modified versions for this example are renamed with the suffix Globus; for example, SubmitGlobus is the submission script and GenerateDrawsGlobus is the program generating random numbers. See “GenerateDrawsGlobus.C” on page 352 and “SubmitGlobus script” on page 353.

The sample broker developed in “ITSO broker” on page 327 will be used as in the previous example to obtain execution host names.

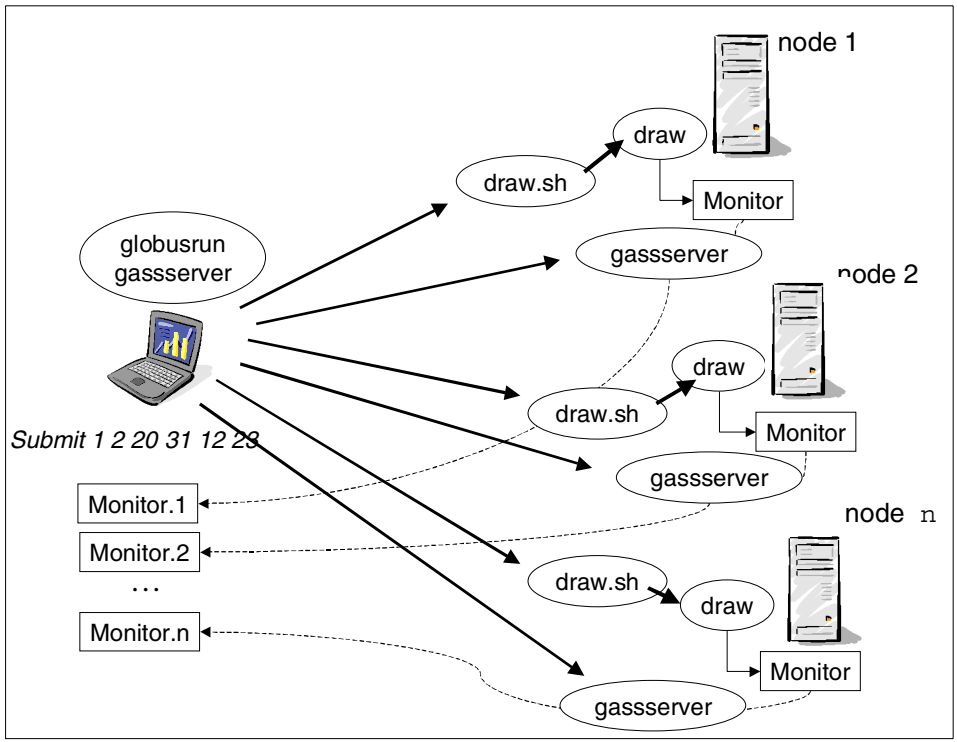


Figure 9-2 Lottery example using Globus commands

Submitting the jobs

The tag ressourceManagerContact will be used for the RSL string to specify a multi-request query. Each resourceManagerContact indicates one execution node. The structure of the query is:

```
+  (&(resourceManagerContact="host1")(executable= ....) )
   (&(resourceManagerContact="host2")(executable= ....) )
   (&(resourceManagerContact="host3")(executable= ....) )
```

globusrun is a blocking shell command that will wait for the completion of the jobs. Consequently, we run **globusrun** in the background so that the submission script (SubmitGlobus in this example) can continue running and acts as a non-blocking command. The **wait** command can be used to wait for the completion of this command. However, in the example it is not really required.

Example 9-3 Building a multi-request query for globusrun

```
echo Jobs submission to the grid
rsl="+"
for i in $loop
```

```
do
   echo ${Nodes[$i]}
   rsl=${rsl}"(&(resourceManagerContact=\"${Nodes[$i]}\")"
   rsl=${rsl}"(executable=\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus.sh)
          (arguments=$TMP $n \"$param\")
          (subjobStartType=loose-barrier)
          (file_stage_in=(\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus
                      GenerateDrawsGlobus.$TMP))
          (file_clean_up=GenerateDrawsGlobus.$TMP)
          (environment=(LD_LIBRARY_PATH \$(GLOBUS_LOCATION)/lib)) )"
done
echo $rsl
#start the commands in the background to be non-blocking
globusrun -s -o "$rsl" &
```

Note: (subjobStartType=loose-barrier) must be used in the RSL commands to avoid premature ending of the sub-jobs that do not terminate at the same time as the others because they run slower.

Job submission

The script GenerateDrawsGlobus.sh is used to invoke GenerateDrawsGlobus and to perform the simulation on each execution node.

Example 9-4 GenerateDrawsGlobus.sh

```
#First argument is the hostname
#Second arguement is the number of draws to simulate
#third argument is the draw to test (“1 21 32 12 24 43 45”)
chmod +x ~/GenerateDrawsGlobus.$1
~/GenerateDrawsGlobus.$1 $1 $2 | grep “$3” && echo GOT IT on $HOSTNAME
```

GenerateDrawsGlobus.sh is used as an intermediary to start the computation instead of directly invoking GenerateDrawsGlobus. It is needed because it performs three actions that cannot be described in one RSL string:

- It make GenerateDrawsGlobus executable.
- It invokes GenerateDrawsGlobus.
- It pipes the GenerateDrawsGlobus output through grep.

We use the local GASS server that is started with globusrun and the -s option, to perform the movement of the executables GenerateDrawsGlobus.sh and GenerateDrawsGlobus. Both files are staged in the execution nodes (see the SubmitGlobus script):

```
(executable=\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus.sh)
(file_stage_in=(\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus
               GenerateDrawsGlobus.$TMP)
```

TMP is used for the same reason, as in the previous example, to avoid conflicts between jobs submitted from different nodes so that they do not work on the same files. TMP equals $HOSTNAME, actually the submission node host name. The process ID of the SubmitGlobus script could be used to add more granularity and avoid conflicts between jobs submitted from the same host and from different users or with different parameters.

If one good result is found, the stdout output is redirected locally from the execution node to the submission node, and will appear during the execution.

GenerateDrawsGlobus generates random draws. It is slightly different from the previous GenerateDraws program in that it writes the Monitor file under a different file name: Monitor.<submission node hostname>.

```
filename="Monitor.";
filename.append(argv[1]);
OutputFileMonitor.open(filename.c_str());
```

The <submission node hostname> is actually passed as a parameter (see “GenerateDrawsGlobus.C” on page 352) by the GenerateDrawsGlobus.sh that itself receives these parameters from the RSL command:

```
In the SubmitGlobus script
(executable=GenerateDraws.sh)(arguments=$TMP $n \"$param\")
In GenerateDraws.sh script:
~/GenerateDrawsGlobus.$1 $1 $2 | grep “$3” && echo GOT IT on $HOSTNAME
```

For a job submitted from m0.itso-maya.com to t1.itso-tupi.com, the RSL string is:

```
&(resourceManagerContact="c1.itso-cherokee.com")
(executable=$(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlobus.
sh)
(arguments=m0.itso-maya.com 300000 "2 3 6 7 8 20 45 55")
(subjobStartType=loose-barrier)
(file_stage_in=($(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlo
bus GenerateDrawsGlobus.m0.itso-maya.com))
(file_clean_up=GenerateDrawsGlobus.m0.itso-maya.com)
(environment=(LD_LIBRARY_PATH $(GLOBUS_LOCATION)/lib)) )G
```

GASS servers on the execution nodes

A GASS server is started on all execution nodes. We use **globusrun** to start this server. The command **globusrun** is submitted in the background so that it is non-blocking in the script. The URL of the GASS server is written in the file gass-server.# in the current directory, where # is an index used in the script to refer to the execution nodes.

Example 9-5 Starting a remote GASS server using globusrun

```
for i in $loop
```

```
do
   rsl='&(executable=$(GLOBUS_LOCATION)/bin/globus-gass-server)(arguments=-c -t
-r)(environment=(LD_LIBRARY_PATH
$(GLOBUS_LOCATION)/lib))(file_clean_up=Monitor.'"$TMP)"
   globusrun -o -r ${Nodes[$i]} "$rsl" > gass-server.$i &
done
```

As the GASS server is not started immediately, we test the size of the file gass-server.# before trying to communicate with this server. This size will actually remain null as long as the GASS server started remotely, and has not returned the URL on which it will listen.

```
if [ -s gass-server.$i ]
then
   contact=$(cat gass-server.$i)
   globus-url-copy $contact/~/Monitor.$TMP file://$PWD/Monitor.$i
   status=$(cat Monitor.$i)
   statussum=$((  $status + $statussum   ))
   echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
fi
```

Finally, the GASS servers are shut down at the end of the script by using **globus-gass-server-shutdown**, as shown in Example 9-6.

Example 9-6 Shutting down the remote GASS servers

```
for i in $loop
do
   contact=$(cat gass-server.$i)
   globus-gass-server-shutdown $contact
done
```

Monitoring

globus-url-copy is used to copy the Monitor files created by **GenerateDrawsGlobus**. The Monitor files are not redirected to the submission node via GASS because **GenerateDrawsGlobus** keeps writing in the file and that would cause a lot of network traffic.

Example 9-7 Monitoring

```
echo Monitoring
rm -f Monitor.*
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
   sleep 5  #we poll every 5 seconds
   statussum=0
```

```
   for i in $loop
   do
      if [ -s gass-server.$i ]
      then
      contact=$(cat gass-server.$i)
           globus-url-copy $contact/~/Monitor.$TMP file://$PWD/Monitor.$i
           status=$(cat Monitor.$i)
           statussum=$((  $status + $statussum   ))
           echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
      fi
   done
   statussum=$((  $statussum  / 100 ))
done
```

Because the remote GASS server may not have started yet, we check the size of the gass-server.$i file. If empty, that means that no URL has been returned yet and therefore the GASS server has not yet started.

As in Example 9-7 on page 258, the for loop scans the content of each Monitor.$i copied from each execution host and displays them every five seconds.

Implementation

Below we discuss the implementation.

Example 9-8 SubmitGlobus script

```
#the script takes the tested draw as parameter
#example: ./Submit 3 4 5 32 34 43
n=300000
NodesNumber=8

#temporary filename used by by GenerateDrawsGlobus
#to monitor the job
#we can also use the process id to increase the granularity
TMP=$HOSTNAME

i=0
#the loop variable is used is all the “for” loops
#the format is 1 2 3 4 .... n
loop=””
# use here the broker developped for the publication
# see chapter 8 (mds executable)
for node in $(mds $NodesNumber | xargs)
do
   Nodes[$i]=$node
   loop=${loop}” “${i}
   i=$(( $i + 1 ))
done
```

```
echo The number of draws tested is $n
a=$*
#sort the numbers in the specified draw
# 2 45 23 12 32 43 becomes 2 12 23 32 43 45 so that we could use
# grep to test this draw and the ouput of the draw programs.
param=$(echo $a | tr " " "\n" | sort -n | xargs )

#Start the gass server on each nodes
# clean up the Monitoring file when leaving
for i in $loop
do
   rsl='&(executable=$(GLOBUS_LOCATION)/bin/globus-gass-server)(arguments=-c -t
-r)(environment=(LD_LIBRARY_PATH
$(GLOBUS_LOCATION)/lib))(file_clean_up=Monitor.'"$TMP)"
   globusrun -o -r ${Nodes[$i]} "$rsl" > gass-server.$i &
done
#file should be made executable
#on all the execution nodes
echo Jobs submission to the grid
rsl="+"
for i in $loop
do
   echo ${Nodes[$i]}
   rsl=${rsl}"(&(resourceManagerContact=\"${Nodes[$i]}\")"

rsl=${rsl}"(executable=\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus.sh)(argum
ents=$TMP $n
\"$param\")(subjobStartType=loose-barrier)(file_stage_in=(\$(GLOBUSRUN_GASS_URL
)$PWD/GenerateDrawsGlobus
GenerateDrawsGlobus.$TMP))(file_clean_up=GenerateDrawsGlobus.$TMP)(environment=
(LD_LIBRARY_PATH \$(GLOBUS_LOCATION)/lib)) )"
done
echo $rsl
globusrun -s -o "$rsl" &
#for monitoring, we copy locally the Monitor files
# created on each compute nodes.  This file content the
# percentage of tested draws.  Each files is suffixes by
# the nodes number.  $statusnum is actually the sum of all
# the percentage (Monitor files) devided by 100.  When it
# equals the number of nodes, that means that we have finished

echo Monitoring
rm -f Monitor.*
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
```

```
   sleep 5  #we poll every 5 seconds
   statussum=0
   for i in $loop
   do
      if [ -s gass-server.$i ]
      then
      contact=$(cat gass-server.$i)
           globus-url-copy $contact/~/Monitor.$TMP file://$PWD/Monitor.$i
           status=$(cat Monitor.$i)
           statussum=$((  $status + $statussum   ))
           echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
      fi
   done
   statussum=$((  $statussum  / 100 ))
done

#Stop the gassserver
for i in $loop
do
   contact=$(cat gass-server.$i)
   globus-gass-server-shutdown $contact
done
```

mds is the broker executable described in “Broker example” on page 127. It must be in the PATH because it is invoked by the SubmitGlobus script.

For a a short computation on three nodes the result is:

```
[globus@m0 other]$ ./SubmitGlobus 2 3 45 6 7 8 20
The number of draws tested is 100000
Jobs submission to the grid
d2.itso-dakota.com
c2.itso-cherokee.com
c1.itso-cherokee.com
+(&(resourceManagerContact="d2.itso-dakota.com")(executable=$(GLOBUSRUN_GASS_UR
L)/home/globus/JYCode/other/GenerateDrawsGlobus.sh)(arguments=m0.itso-maya.com
100000 "2 3 6 7 8 20 45") (subjobStartType=loose-barrier)
(file_stage_in=($(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlo
bus GenerateDrawsGlobus.m0.itso-maya.com))
(file_clean_up=GenerateDrawsGlobus.m0.itso-maya.com)
(environment=(LD_LIBRARY_PATH $(GLOBUS_LOCATION)/lib)) )
(&(resourceManagerContact="c2.itso-cherokee.com")
(executable=$(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlobus.
sh) (arguments=m0.itso-maya.com 100000 "2 3 6 7 8 20 45")
(subjobStartType=loose-barrier)
(file_stage_in=($(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlo
bus GenerateDrawsGlobus.m0.itso-maya.com))
(file_clean_up=GenerateDrawsGlobus.m0.itso-maya.com)
(environment=(LD_LIBRARY_PATH $(GLOBUS_LOCATION)/lib)) )
```

```
(&(resourceManagerContact="c1.itso-cherokee.com")
(executable=$(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlobus.
sh) (arguments=m0.itso-maya.com 100000 "2 3 6 7 8 20 45")
(subjobStartType=loose-barrier)
(file_stage_in=($(GLOBUSRUN_GASS_URL)/home/globus/JYCode/other/GenerateDrawsGlo
bus GenerateDrawsGlobus.m0.itso-maya.com))
(file_clean_up=GenerateDrawsGlobus.m0.itso-maya.com)
(environment=(LD_LIBRARY_PATH $(GLOBUS_LOCATION)/lib)) )
Monitoring


d2.itso-dakota.com:Monitor 71 %
c2.itso-cherokee.com:Monitor 66 %
c1.itso-cherokee.com:Monitor 61 %

d2.itso-dakota.com:Monitor 100 %
c2.itso-cherokee.com:Monitor 100 %
c1.itso-cherokee.com:Monitor 96 %

d2.itso-dakota.com:Monitor 100 %
c2.itso-cherokee.com:Monitor 100 %
c1.itso-cherokee.com:Monitor 100 %
```

9.2 Small Blue example

This example shows an example of how to distribute a function across a Grid infrastructure based on the Globus Toolkit 2.2. Note that for readability reasons, not all the needed error checking is done for every Globus Toolkit 2.2 API call.

The purpose of the game (called Puissance 4, in French) is to align four chips to win. In this example, a simple artificial intelligence machine plays against a human.

The artificial intelligence is implemented in the GAME.C program available in “GAME Class” on page 337 and works in the following ways:

- It evaluates the value of each position from the first column to the eighth.
- For each position, it also evaluates the next positions that the adversary could possibly play and reevaluates its tested position accordingly.
- When all positions are evaluated, the best move is chosen.

The standalone or non-gridified version of the game is available in “SmallBlue.C (standalone version)” on page 331. The algorithm as well as the GAME class is not studied in the publication. The GAME class provides methods to display the

game, to check if someone has won, to play the position decided by the players, and to test if a player can play a specific column.

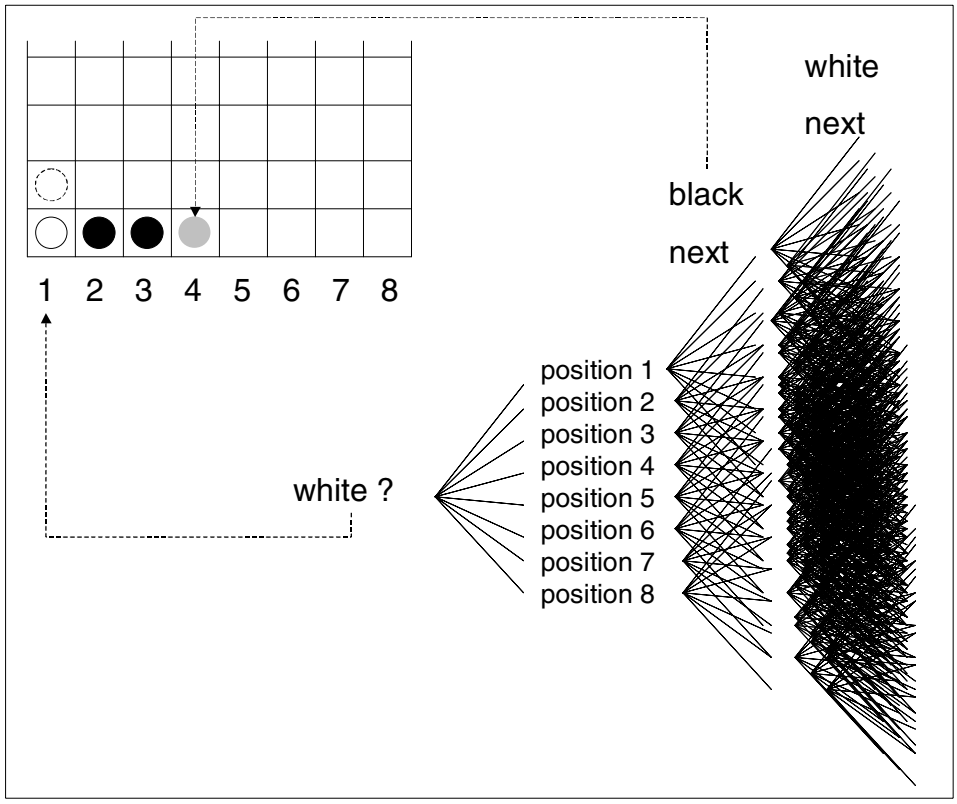


Figure 9-3 Problem suitable for Grid enablement

In the extract of the SmallBlue.C (Example 9-9 on page 264) source code, we can see that:

- The evaluation of a position is performed via the Simulate() function called from the main() program.
- The SmallBlue application uses the object Current of type GAME to store the game data. This data is used to perform the evaluation of a position. The evaluation of a position is performed in the function Simulate() that takes two parameters: The tested position and the game data. Simulate()returns the value of the evaluation. This function is called for each column in main().
- The Value() method of the GAME class is used to evaluate the value of a position. It takes the tested column as a parameter as well as the player (BLACK/WHITE) who is playing the column. This method is called in Simulate().

Example 9-9 Standalone version

```
int Simulate(GAME newgame, int col) {
   int l=0,s;
   int start=newgame.Value(col,WHITE);
   newgame.Play(col,WHITE);
   l=0;
   for(int k=1;k!=XSIZE+1;k++) {
      s=newgame.Value(k,BLACK);
      if (s>l)
         l=s;
   };
   start-=l;
   return start;
}

main() {
   //************************** Start **************************/
   GAME Current(XSIZE,YSIZE);
   int s,l,k,toplay;
   char c[2];
   while (true) {
      Current.Display();
      do {
         cout << "?";
         cin >> c;
         c[1]='\0';
         l=atoi(c);
      } while ((l<1) || (l>XSIZE) || !Current.CanPlay(l) );
      Current.Play(l,BLACK);
      if (Current.HasWon(l,BLACK)) {
         Current.Display();
         exit(1);
      };
   //************************** Simulation ***************************/
      l=-100000;
      for(k=1;k!=XSIZE+1;k++) {
         if (Current.CanPlay(k)) {
   //************************** call Simulate ***************************/
            s=Simulate(Current,k);
            if (s>l) {
               l=s;
               toplay=k;
            };
         };
      };
      if (l==-100000) {
         cout << "NULL" << endl;
         exit(1);
```

```
      };
      Current.Play(toplay,WHITE);
      if (Current.HasWon(toplay,WHITE)) {
         Current.Display();
         exit(1);
      };
   };
};
```

The purpose of this example is to gridify the application by executing the Simulate() function on a remote host:

- Each evaluation of a tested column can be executed independently.
- Each evaluation modifies the game data when simulating an attempt so each job needs to have its own copy of the game. This behavior is also present in the function Simulate() where a new object GAME called newgame is created specifically for the evaluation, and used to store successive tested positions. Therefore, the game data must be replicated on all execution nodes.

9.2.1 Gridification

To gridify this application, we will use two programs:

- One called SmallBlueMaster that will submit the job, gather the results, and be the interface with the human player
- One called SmallBlueSlave that will perform the simulation and returns the result to SmallBlueMaster

The source code for these two programs is available in “SmallBlueMaster.C” on page 332 and “SmallBlueSlave.C” on page 336.

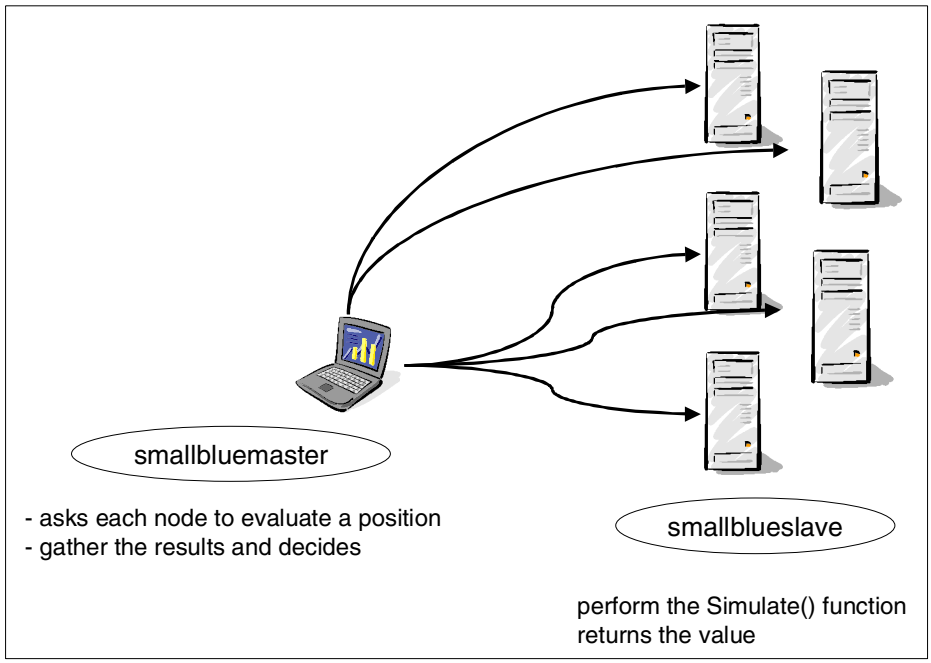


Figure 9-4 Gridified SmallBlue

One problem is that the simulate() function uses two variables and returns one value that needs to be passed to and retrieved from a remote host. We cannot use inter-process communications between the nodes.

A solution is to serialize objects by storing the instance value on disk and use the Globus Toolkit 2.2 data movement functions:

- By using GRAM and GASS systems. The `Current` object will be serialized to disk and transferred to each remote host. The tested position will be passed as an argument to SmallBlueslave, which will load the current value and therefore will recreate the same environment that exists in SmallBlueMaster.
- By using the GRAM and GASS subsystems, all the results of the execution nodes will be output to the same file, eval, on the master node.

Communication is accomplished:

1. By a local GASS server started on the master node and listening on port 10000
2. By the GASS servers started on each execution node by the GRAM, which will map standard input and output to remote files

The following RSL command describes this process:

```
&(executable=smallblueslave) (arguments=<tested column>)
(stdout=https://<masternode>:10000/<localdir>/eval)
(stdin=https://<masternode>:10000/<localdir>/GAME) (count=1)"
```

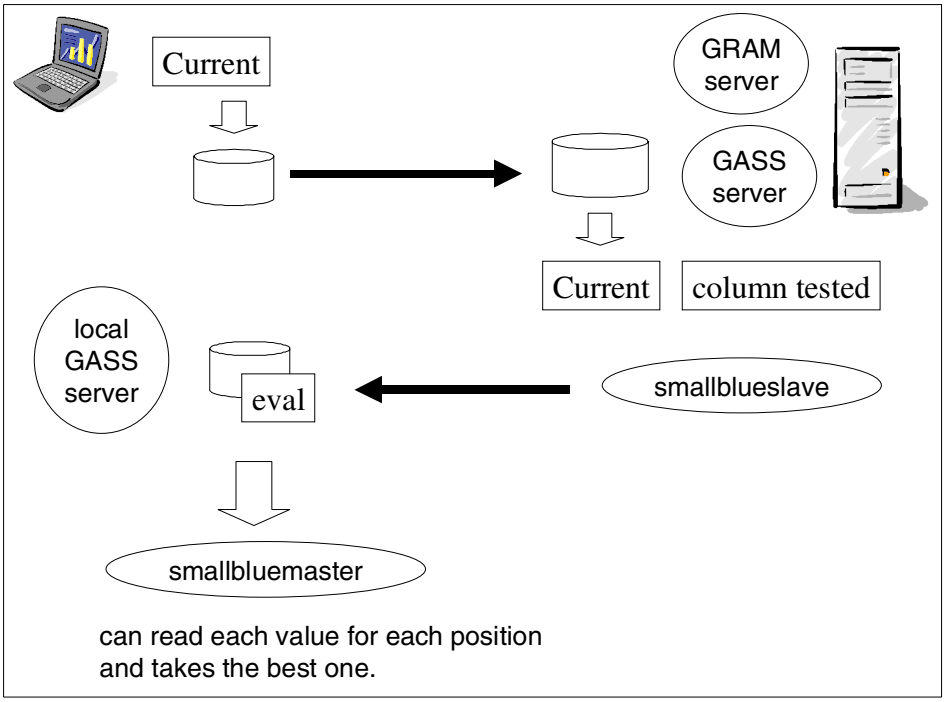


Figure 9-5 How to transfer an object via GRAM and GASS

The local GASS server started by SmallBlueMaster transparently provides access to the eval and GAME file to the remote execution nodes via GRAM and the associated GASS server. It arbitrarily listens on port 10000. Two functions, StartGASSServer() and StopGASSServer(), wrap the Globus Toolkit 2.2 API calls to start and stop the local GASS server. The source code of these functions is available in “itso_gass_server.C” on page 325.

We can see in the SmallBlueSlave code source that it reads and writes only on standard input and output channels (via cin and cout standard iostream objects). It will nevertheless transparently work on remotely stored files thanks to the stdin and stdout keywords in the RSL job description language:

```
(stdout=https://<masternode>:10000/<localdir>/eval)
(stdin=https://<masternode>:10000/<localdir>/GAME)
```

Where:

- stdin is mapped to the GAME file located on the master node and generated by the call to the method ToDisk() of the object Current (object serialization). Another way to proceed would be to use Globus sockets to transfer the serialized object without the need of intermediate files.
- stdout is mapped to the eval file located on the master node. SmallBlueSlave will write the value of a tested position to this file. All the nodes write to eval so all output will be appended to this file.

SmallBlueSlave also needs a parameter (the tested column) that will be passed as a parameter to the program via the (arguments=)expression of the RSL job submission string. Then SmallBlueSlave uses argv[1] to retrieve this parameter.

Finally, we will use the GridFTP protocol (as an example), to transfer the SmallBlueSlave executable to a remote host. The transfer is achieved via the ITSO_GLOBUS_FTP_CLIENT class and its transfer() method. The source code is available in “itso_globus_ftp_client.C” on page 313.

9.2.2 Implementation

We will use three C++ classes that wrap Globus C calls:

- ITSO_CB will be used as generic callback type for all globus functions that need a callback. Note that these objects are always called from a static C function whose one argument is the object itself. ITSO_CB implements the mutex, condition variables synchronization mechanism always used with the Globus Toolkit 2 non-blocking or asynchronous functions. ITSO_GRAM_JOB and ITSO_GLOBUS_FTP_CLIENT both derive from ITSO_CB. See “ITSO_CB” on page 315 and its explanation in “Callbacks” on page 109.
- ITSO_GRAM_JOB will be used to submit a job. See “ITSO_GRAM_JOB” on page 316.
- ITSO_GLOBUS_FTP_CLIENT, which is a wrapper class to the C globus client ftp functions, will perform a transfer from a file stored in a storage server to a remote URL. See “ITSO_GLOBUS_FTP_CLIENT” on page 311.

The complete source code of the two programs is available in “SmallBlue example” on page 331.

StartGASSServer() and StopGASSServer() are the two functions respectively used to start and stop the local GASS server that will retrieve the result of the evaluation nodes. The source code is provided in “StartGASSServer() and StopGASSServer()” on page 324

To copy the file SmallBlue in parallel to each remote host, we use the GridFTP protocol via the ITSO_GLOBUS_FTP_CLIENT:

```
vector<ITSO_GLOBUS_FTP_CLIENT*> transfer;
globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);
string dst;
for(i=0;i!=8;i++) {
   cout << node[i] << endl;
   dst="gsiftp://"+node[i]+"/~/SmallBlueSlave";
   transfer.push_back(new ITSO_GLOBUS_FTP_CLIENT("SmallBlueSlave",
                      const_cast<char*>(dst.c_str())));
};
for(i=0;i!=8;i++)
   transfer[i]->StartTransfer();
for(i=0;i!=8;i++)
   transfer[i]->Wait();
globus_module_deactivate(GLOBUS_FTP_CLIENT_MODULE);
```

We also need to make **SmallBlue** executable on the remote hosts because the file copied by GridFTP is copied as a plain file and not as an executable.

```
for(i=0;i!=8;i++) {
   rsl_req = "&(executable=/bin/chmod) (count=1) (arguments= \"+x\"
               SmallBlueSlave)";
   if ( job[i]->Submit(node[i],rsl_req))
      exit(1);
}
for(i=0;i!=8;i++)
   job[i]->Wait();
```

Example 9-10 SmallBlue Gridfication - Initialization - SmallBlueMaster.C

```
main() {
   //Start a GASS server locally that will listen on 10000 port
   //all the results of the evualtion.  We will stop it at the end
   //It cannot be defined as a standalone class because the static callback
   //does not take any argument.  So it is impossible afterwards in the
   //callback to refer to the object.
   StartGASSServer(10000);

   // ITSO_GRAM_CLIENT does not start the module
   // lets do it
   if (globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE) != GLOBUS_SUCCESS)
   {
      cerr << " Cannot start GRAM module";
      exit(2);
   };

   // the game
   GAME Current(XSIZE,YSIZE);
```

```
   // used to temporary store columns positions, evaluation results
   int s,l,k,toplay;
   // used to store human inputs
   char c[2];

   // The node vector should be initialized with the value of the nodes
   // what is missing here is the globus calls to the globus MDS server
   // to get these values.  So for the exercise, you can use grid-info-search
   // to find 8 hosts on which you can submit your queries.
   vector<string> node;

   // ask the broker to find 8 nodes
   itso_broker::GetLinuxNodes(node,8);

   // variable used in all for loops
   int i;

   // Here we want test the existence of the file as there is
   // no such checking in the ITSO_GLOBUS_FTP_CLIENT class
   FILE*    fd = fopen("SmallBlueSlave","r");
        if(fd == NULL)
   {
              printf("Error opening local smallblueslave");
              exit(2);
   }
   else {
      //that fine, lets go for FTP
      // we can close fd descriptor because a new one
      // will be opened for each ITSO_GLOBUS_FTP_CLIENT object
      fclose(fd);
      // the ITSO_CB callback object is used to determine
      // when the transfer has been completed
      vector<ITSO_GLOBUS_FTP_CLIENT*> transfer;
      //never forget to activate the Globus module you want to use
      globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);

      // 8 transfer, let create 8 locks
      string dst;
      for(i=0;i!=8;i++) {
         cout << node[i] << endl;
         dst="gsiftp://"+node[i]+"/~/SmallBlueSlave";
         transfer.push_back(new
ITSO_GLOBUS_FTP_CLIENT("SmallBlueSlave",const_cast<char*>(dst.c_str())));
      };
      // Let s begin the transfer in parallel (in asynchronous mode)
      for(i=0;i!=8;i++)
         transfer[i]->StartTransfer();
      // Let wait for the end of all of them
      for(i=0;i!=8;i++)
```

```
         transfer[i]->Wait();
      globus_module_deactivate(GLOBUS_FTP_CLIENT_MODULE);
   };

   // get the hostname using the globus shell function
   // instead of POSIX system calls.
   char hostname[MAXHOSTNAMELEN];
   globus_libc_gethostname(hostname, MAXHOSTNAMELEN);

   // used to store the RSL commands.
   string rsl_req;

   //create all the jobs objects that will be used to submit the
   //requests to the nodes.  We use a vector to store them.
   vector<ITSO_GRAM_JOB*> job;
   for(i=0;i!=8;i++) {
      job.push_back(new ITSO_GRAM_JOB);
   };

   // By using gridftp SmallBlueSlave is copied onto the rmeote hosts
   // as a plain file.  needs to chmod +x to make it executable
   // otherwise the job submission will fail
   cout << "chmod +x on the remote hosts to make SmallBlueSlave executable" <<
endl;
   for(i=0;i!=8;i++) {
      rsl_req = "&(executable=/bin/chmod) (count=1) (arguments= \"+x\"
SmallBlueSlave)";
      if ( job[i]->Submit(node[i],rsl_req))
         exit(1);
   }
   for(i=0;i!=8;i++)
      job[i]->Wait();
   //finished stop everything
   StopGASSServer();
}
```

The game between the two players is run in an infinite loop in which we repeatedly:

- Serialize the current game:

```
Current.ToDisk("GAME");
```

- Submit the calculation of the value for each column:

```
unlink("eval"); //remove the eval file
for(i=0;i!=8;i++) {
   cout << "submission on " << node[i] << endl;;
   char tmpc[2];
   sprintf(tmpc,"%d",i);
```

```
      // build the RSL commands
      rsl_req = "&(executable=SmallBlueSlave) (arguments=";
      rsl_req+= tmpc[0];
      rsl_req+= ") (stdout=https://";
      rsl_req +=hostname;
      rsl_req +=":10000";
      rsl_req +=get_current_dir_name();
      rsl_req +="/eval) (stdin=https://";
      rsl_req +=hostname;
      rsl_req +=":10000";
      rsl_req +=get_current_dir_name();
      rsl_req +="/GAME) (count=1)";
      // submit it to the GRAM
      if (Current.CanPlay(i))
         if (job[i]->Submit(node[i],rsl_req))
            exit(1);
   };
   // And Wait
   for(i=0;i!=8;i++)
      if (Current.CanPlay(i))
         job[i]->Wait();
```

- Retrieve the results:

```
ifstream results("eval");
while (!results.eof()) {
   results >> k >> s;
   // get the best one
   if (s>l) {
      l=s;      // store its value
      toplay=k; //remember the column to play
   };
};
results.close();
```

Finally, we exit the loop when one of the two players has won, by using the HasWon() method of the `Current` object:

```
if (Current.HasWon(l,BLACK)) {
   Current.Display();
   break;
```

Example 9-11 SmallBlue Gridification - SmallBlueMaster.C

```
.....................................
   while (true) {
      Current.Display();
      do {
         cout << "?";
         cin >> c;
         c[1]='\0';
```

```
   l=atoi(c);
} while ((l<1) || (l>XSIZE) || !Current.CanPlay(l) );
Current.Play(l,BLACK);
if (Current.HasWon(l,BLACK)) {
   Current.Display();
   break;
};
// Serialize to disk the Current variable
// so that it could be used by the GRAM
// subsystem and transferred on the remote execution
// nodes
Current.ToDisk("GAME");

Current.Display();
cout << endl;

// remove eval file for each new jobs submission
// otherwise results will be appended to the same files
unlink("eval");

for(i=0;i!=8;i++) {
   cout << "submission on " << node[i] << endl;;
   char tmpc[2];
   sprintf(tmpc,"%d",i);
   // build the RSL commands
   rsl_req = "&(executable=SmallBlueSlave) (arguments=";
   rsl_req+= tmpc[0];
   rsl_req+= ") (stdout=https://";
   rsl_req +=hostname;
   rsl_req +=":10001";
   rsl_req +=get_current_dir_name();
   rsl_req +="/eval) (stdin=https://";
   rsl_req +=hostname;
   rsl_req +=":10001";
   rsl_req +=get_current_dir_name();
   rsl_req +="/GAME) (count=1)";
   // submit it to the GRAM
   if (Current.CanPlay(i))
      if (job[i]->Submit(node[i],rsl_req))
         exit(1);
};
// And Wait
for(i=0;i!=8;i++)
   if (Current.CanPlay(i))
      job[i]->Wait();

// worse case :-)
l=-100000;
```

```
      //Here we are reading the eval files.  All the jobs
      //has been completed so we should have all the results
      //in the eval file
      ifstream results("eval");
      while (!results.eof()) {
         results >> k >> s;
         // get the best one
         if (s>l) {
            l=s;      // store its value
            toplay=k; //remember the column to play
         };
      };
      results.close();

      // nothing in the file, that means we cannot play
      // so it is NULL
      if (l==-100000) {
         cout << "NULL" << endl;
         break;
      };

      // AI plays here and checks if it won
      Current.Play(toplay,WHITE);
      if (Current.HasWon(toplay,WHITE)) {
         Current.Display();
         break;
      };
   };
.....................................
```

The slave code is small. GAME.C implements the game artificial intelligence.

The slave begins to read the serialized object from standard input (actually mapped to GAME file on the master node):

```
Current.FromDisk();
```

Then the slave only tests the eight positions that can be played by the adversary. It writes the result of the evaluation to standard output. The GASS server started by GRAM actually maps this standard output to the file eval on the submission node.

Example 9-12 SmallBlueSlave.C

```
#include <iostream>
using namespace std;
#include "GAME.C"

main(int arc, char** argv) {
```

```
   GAME Current(XSIZE,YSIZE);
   //load the Current object from the disk
   //this object was copied from the submission node
   Current.FromDisk();
   //which column should we simulate ?
   int col=atoi(argv[1]);
   int start=Current.Value(col,WHITE);
   Current.Play(col,WHITE);


   int l=0,s,k;
   for(k=1;k!=XSIZE+1;k++) {
      s=Current.Value(k,BLACK);
      if (s>l)
         l=s;
   };
   start-=l;

   // send back the information to the server
   cout << col << " " << start << endl;
};
```

9.2.3 Compilation

First generate the appropriate globus makefile header that will be later included in the Makefile. Use **globus-makefile-header** and specify all the needed globus modules.

```
globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist
globus_ftp_client globus_ftp_control globus_gram_job globus_common
globus_gram_client globus_gass_server_ez > globus_header
```

Compile with the following Makefile:

```
make -f MakefileSmallBlue
```

Example 9-13 MakefileSmallBlue

```
globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist
globus_ftp_client globus_ftp_control globus_gram_job globus_common
globus_gram_client globus_gass_server_ez > globus_header

include globus_header

all:  SmallBlueSlave SmallBlueMaster SmallBlue

%.o: %.C
   g++ -c $(GLOBUS_CPPFLAGS) $< -o $@
```

```
SmallBlue:SmallBlue.o GAME.o
   g++ -o $@ -g $^

SmallBlueSlave:SmallBlueSlave.o GAME.o
   g++ -o $@ -g $^

SmallBlueMaster: GAME.o SmallBlueMaster.o itso_gram_job.o itso_cb.o
itso_globus_ftp_client.o itso_gass_server.o
   g++ -g -o $@ $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS)
```

9.2.4 Execution

Issue **grid-proxy-init** to acquire a valid credential in the grid.

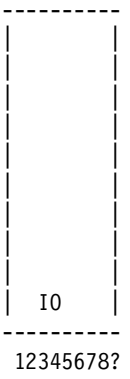
Start SmallBlueMaster and enter the column number you want to play:

```
[globus@m0 JYCode]$ ./SmallBlueMaster
we are listening on https://m0.itso-maya.com:10000
chmod +x on the remote hosts to make SmallBlueSlave executable
Contact on the server https://t1.itso-tupi.com:34475/16083/1047519201/
Contact on the server https://t2.itso-tupi.com:33326/6614/1047519203/
Contact on the server https://t0.itso-tupi.com:58412/7839/1047519203/
Contact on the server https://t3.itso-tupi.com:55107/29288/1047519236/
Contact on the server https://t2.itso-tupi.com:33328/6615/1047519203/
Job Finished on: https://t1.itso-tupi.com:34475/16083/1047519201/
Contact on the server https://t0.itso-tupi.com:58414/7840/1047519203/
Contact on the server https://t1.itso-tupi.com:34478/16085/1047519201/
Contact on the server https://t3.itso-tupi.com:55110/29289/1047519237/
Job Finished on: https://t2.itso-tupi.com:33328/6615/1047519203/
Job Finished on: https://t2.itso-tupi.com:33326/6614/1047519203/
Job Finished on: https://t1.itso-tupi.com:34478/16085/1047519201/
Job Finished on: https://t0.itso-tupi.com:58412/7839/1047519203/
Job Finished on: https://t0.itso-tupi.com:58414/7840/1047519203/
Job Finished on: https://t3.itso-tupi.com:55107/29288/1047519236/
Job Finished on: https://t3.itso-tupi.com:55110/29289/1047519237/


----------
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
```

```
----------
 1234567?3


----------
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|        |
|  I     |
----------
 12345678
submission on t0
submission on t1
submission on t2
submission on t3
submission on t0
submission on t1
submission on t2
submission on t3
Contact on the server https://t2.itso-tupi.com:33335/6644/1047519211/
Contact on the server https://t1.itso-tupi.com:34489/16114/1047519209/
Contact on the server https://t0.itso-tupi.com:58421/7869/1047519211/
Staging file in on: https://t2.itso-tupi.com:33335/6644/1047519211/
Staging file in on: https://t1.itso-tupi.com:34489/16114/1047519209/
Staging file in on: https://t0.itso-tupi.com:58421/7869/1047519211/
Contact on the server https://t2.itso-tupi.com:33339/6645/1047519211/
Contact on the server https://t1.itso-tupi.com:34485/16113/1047519208/
Contact on the server https://t0.itso-tupi.com:58424/7870/1047519211/
Staging file in on: https://t1.itso-tupi.com:34485/16113/1047519208/
Staging file in on: https://t2.itso-tupi.com:33339/6645/1047519211/
Contact on the server https://t3.itso-tupi.com:55121/29325/1047519244/
Staging file in on: https://t0.itso-tupi.com:58424/7870/1047519211/
Staging file in on: https://t3.itso-tupi.com:55121/29325/1047519244/
Contact on the server https://t3.itso-tupi.com:55117/29324/1047519244/
Staging file in on: https://t3.itso-tupi.com:55117/29324/1047519244/
Job Finished on: https://t2.itso-tupi.com:33335/6644/1047519211/
Job Finished on: https://t1.itso-tupi.com:34485/16113/1047519208/
Job Finished on: https://t2.itso-tupi.com:33339/6645/1047519211/
Job Finished on: https://t0.itso-tupi.com:58424/7870/1047519211/
Job Finished on: https://t1.itso-tupi.com:34489/16114/1047519209/
Job Finished on: https://t0.itso-tupi.com:58421/7869/1047519211/
Job Finished on: https://t3.itso-tupi.com:55121/29325/1047519244/
Job Finished on: https://t3.itso-tupi.com:55117/29324/1047519244/
```



9.3 Hello World example

Let us consider the following cases of client-server applications or Web applications:

- Video streaming
- Game serving
- File downloading
- Etc.

A classical approach for providing a scalable solution is to distribute the application workload across a set of distributed servers that run the same application: An edge server or network dispatcher or front-end server is the entry point for the application but does not run the application itself. The other servers located on the same LAN handle the workload and answer the client application or client browser.

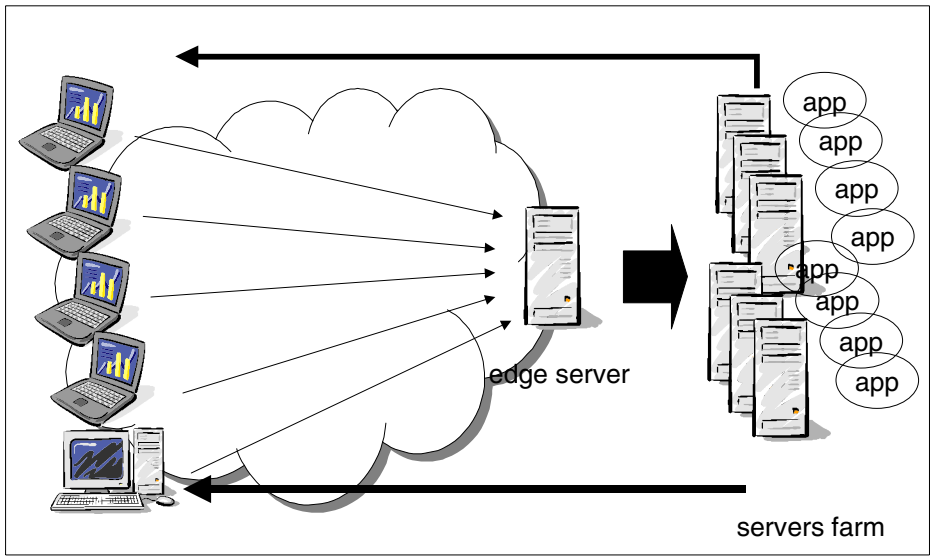


Figure 9-6 Cluster model

A grid approach extends the concept of clustering by enabling the deployment of servers not only on the same LAN but on a WAN infrastructure. The edge server becomes a broker server that will start the applications on remote servers to handle the workload. The broker can use different criteria to manage this workload:

Use the server located at the nearest location from the client.
Use servers according a certain service level agreement with the customer.
Use new servers provided by a resource provider for a limited period of time.
Use a server that has a better network bandwidth.

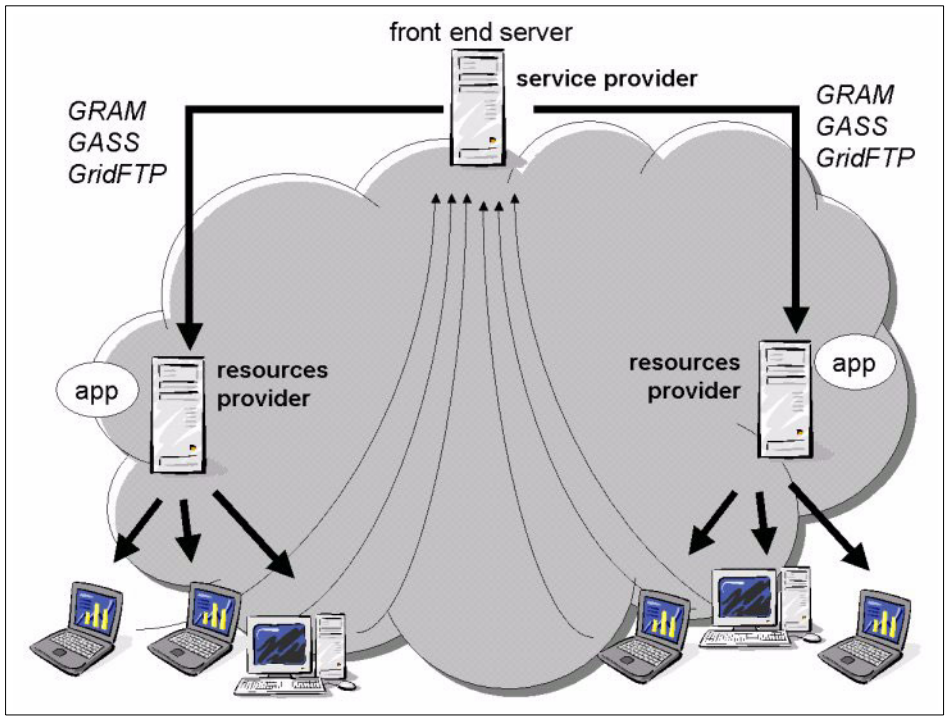


Figure 9-7 Grid model

This extended approach underlines several issues that are managed by the Globus Toolkit 2.2:

- With GSI, the Globus Toolkit 2.2 provides the secure infrastructure needed for an application to be spread across different locations.
- With GridFTP, the Globus Toolkit 2.2 provides a subsystem to easily, efficiently, and securely move data necessary for the applications.
- With GRAM and GASS, the Globus Toolkit 2.2 provides mechanisms to easily and securely start remote applications on distributed resources.

9.3.1 The Hello World application

Let us consider a basic example for such an application. A front-end server waits for client requests. When connected, the client is given back a ticket and an application server IP address of where to connect. The application answers `hello world!` when the client connects to it. The application is started on the "application servers" by the front-end server.

The executable for the client is HelloClient and takes the front-end server host name as a parameter. The source code is provided in “HelloWorld example” on page 341.

The executable for the server itself is HelloServer. The source code is provided in “HelloWorld example” on page 341.

The executable for the front-end server is HelloFrontEnd and starts HelloServer remotely. The source code is provided in “HelloWorld example” on page 341.

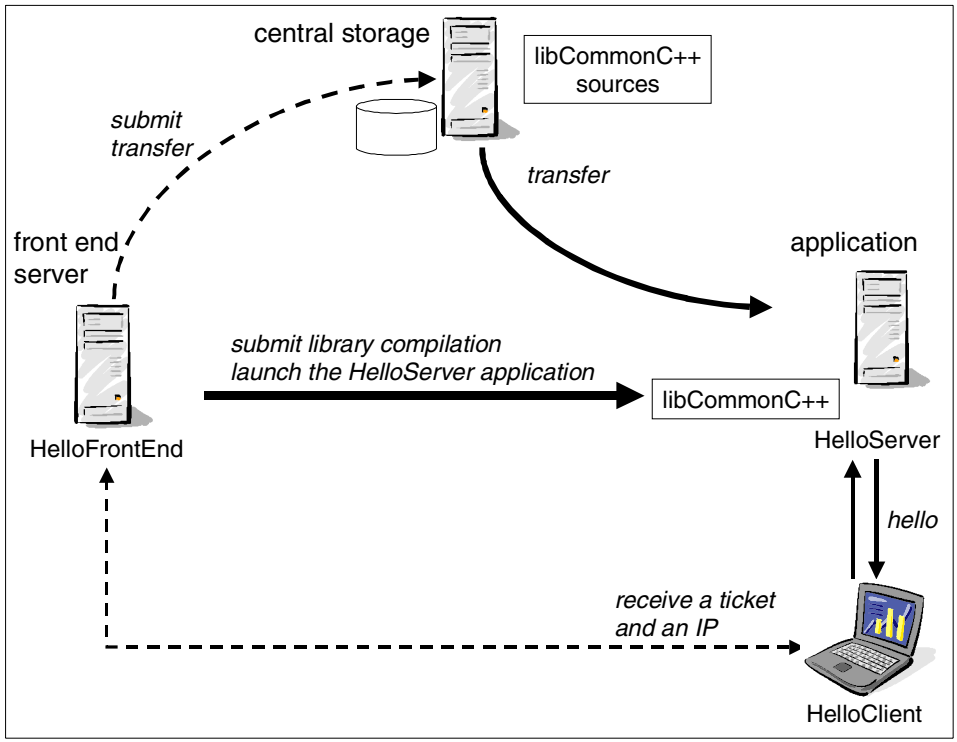


Figure 9-8 Hello World example with dynamic library dependencies issues

9.3.2 Dynamic libraries dependencies

In this example, the application depends on a dynamic library (GNU CommonC++ http://www.gnu.org/software/commonc++/) that is not installed by default on the remote servers. To solve this issue, the front-end server installs at startup, the dynamic library on the remote server by compiling it on this server. This is an interesting solution to avoid runtime issues like libc or libstdc++ dependencies or other library dependencies that Common C++ depends on. This

library can also be recompiled on different architectures with only the copy of the source code to store.

We store the source copy of the library on a storage server (m0 in the example). The script Compile is used to compile the library and install it.

Example 9-14 Compile script

```
################################
# we need to run this script to load
# all correct environment variables.
# The configure script will failed
# otherwise
################################
. ~/.bash_profile

tar -zxf commoncpp2-1.0.8.tar.gz
[ -d tmp ] || mkdir tmp
cd commoncpp2-1.0.8
########################################
# We use $HOME instead of ~ here because
# make install will failed to create
# directory otherwise
########################################
./configure --prefix=$HOME/tmp && make install
######################
#Stay clean
######################
rm -fr commoncpp2-1.0.8
```

The dynamic library is installed in ~/tmp/lib on the remote host. We use the LD_LIBRARY_PATH variable under the Linux operating system to specify the location of the dynamic libraries to the HelloServer executable. This way, we are not intrusive to the remote system (we do not have to be root and install a binary on the system) and this library will not affect applications other than ours.

LD_LIBRARY_PATH is set automatically during the GRAM invocation by using an environment declaration in the RSL string:

```
&(executable=https://m0.itso-maya.com:20000/home/globus/HelloServer)
(environment=(LD_LIBRARY_PATH $(HOME)/tmp/lib) ) (count=1)
(arguments=1804289383 t0)
```

The CommonC++ library source is stored locally in /tmp in m0. We use the ITSO_GASS_TRANSFER class to transfer the file from one GridFTP server to the remote server. The code for the ITSO_GASS_TRANSFER class is provided in “ITSO_GASS_TRANSFER” on page 306 and is based on the globus-url-copy.c code of the Globus Toolkit 2.2. This class provides the member

transfer() between two objects of type GLOBUS_URL. GLOBUS_URL is defined in “ITSO_GASS_TRANSFER” on page 306 and is just a C++ wrapper to the globus_url_t C type. The member setURL() sets up the object by providing a valid URL like gsiftp://m0/tmp/test.

The Transfer() method is a C++ wrapper around the Globus asynchronous call to transfer the file from the first URL to the second. It is non-blocking and the program must call the method Wait() to wait for the completion of the transfer.

GRAM is used to submit the compilation of the GNU CommonC++ library and the application startup. The library is transferred and installed remotely from a remote storage server (m0 in the source code example) to the application node (t0 in the source code example) in the tmp directory created in the home directory of the user under which the Compile script is executed.

A GASS server is started locally on the front-end server and listens on port 20000. It used to copy the script **`Compile`** and the executable **`HelloServer`** from the front-end server to the application server.

Example 9-15 CommonC++ library copy and compilation - HelloFrontEnd.C

```
main() {
   // We start here the GASS server that will be used:
   // - to transfer the Compile script to the application nodes
   //  to perform the library compilation
   // - to transfer the HelloServer used on the remote hosts to
   // manage clients requests
   // The GASS server arbitraly listens on port 20000
   StartGASSServer(20000);
   // get the hostname using the globus function
   globus_libc_gethostname(hostname, MAXHOSTNAMELEN);

   // ITSO_GRAM_CLIENT does not start the module
   // lets do it
   if (globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE) != GLOBUS_SUCCESS)
   {
      cerr << " Cannot start GRAM module";
      exit(2);
   };

   string node;

   // we use a fixed address for this simple example but
   // nodes=getNodes(SLA, MDS, Workload, ...) in a more complex example
   // we should get here the list of nodes where the application is
   // supposed to run.
   node = "t0";
```

```
// variable used in all for loops
int i;

// Here we want test the existence of the file as there is
// no such checking in the ITSO_GLOBUS_FTP_CLIENT class
FILE*    fd = fopen("commoncpp2-1.0.8.tar.gz","r");
     if(fd == NULL)
{
           printf("Error opening commoncpp2-1.0.8.tar.gz file");
           exit(2);
}
else
    {
   //that fine, lets go for FTP
   // we can close fd descriptor because a new one
   // will be opened for each ITSO_GLOBUS_FTP_CLIENT object
   fclose(fd);
   //never forget to activate the Globus module you want to use
   globus_module_activate(GLOBUS_GASS_COPY_MODULE);
   //************************************************************
   // In this section we perform the transfer of the dynamic library
   // to the "application server".
   // We use the ITSO_GLOBUS_FTP_CLIENT class to do the task
   // Note that we use the local directory to perform the task.
   // In a real case example, a storage server would be used
   // instead of a local directory =>
   // globus-url-copy gsiftp://storage_server/commoncpp2-1.0.8.tar.gz
   //       gsiftp://application_server/commoncpp2-1.0.8.tar.gz
   //************************************************************
   GLOBUS_URL source,destination;
   source.setURL("gsiftp://m0/tmp/commoncpp2-1.0.8.tar.gz");

   string dst;
   dst="gsiftp://"+node+"/~/commoncpp2-1.0.8.tar.gz";
   destination.setURL(dst);
   globus_module_activate(GLOBUS_GASS_COPY_MODULE);
   ITSO_GASS_TRANSFER transfer;
   transfer.Transfer(source,destination);
   globus_module_deactivate(GLOBUS_GASS_COPY_MODULE);

   //************************************************************
   // In this section we submit the compilation of the
   // dynamic libray.  The script Compile that must be in the
   // current directory is transferred to the remote host
   // and executed.  The result is the installation
   // of the CommonC++ toolkit in the tmp directory of the
   // user under which the script was executed.
   // (globus in the lab environement of the redbook)
   //************************************************************
```

```
// used to store the RSL commands.
string rsl_req;

ITSO_GRAM_JOB job;

cout << "library compilation on " << node << endl;;
rsl_req = "&(executable=https://";
rsl_req +=hostname;
rsl_req += ":20000";
rsl_req +=get_current_dir_name();
     rsl_req += "/Compile) (count=1)";
// submit it to the GRAM
cout << rsl_req << endl;
//job.Submit(node,rsl_req);
//job.Wait();
//************************************************************

// ticket=getTicket(time, IP address, SLA, ...) in a
// real production case
ticket=random();

// node=getNodes(SLA, MDS, Workload, Application Type, ...)
// in a real production case
node="t0";
```

9.3.3 Starting the application by the resource provider

The front-end server also submits another job to start HelloServer on the application server. HelloServer listens on port 4097 and uses the ticket to authenticate the connection. In the example, the ticket is fixed at the application startup, but for a real production environment other parameters like time, IP address, public/private keys, and so on, may be taken into account to securely authenticate a connection from a client.

The server is arbitrarily started on node t0. Other parameters like MDS information, service level agreements with the customer, workload, resources provider agreements, and such can be used here to determine where the application should be started.

Example 9-16 Front-end server starts the HelloServer on remote node - HelloFrontEnd.C

```
// ticket=getTicket(time, IP address, SLA, ...) in a
// real production case
ticket=random();

// node=getNodes(SLA, MDS, Workload, Application Type, ...)
// in a real production case
```

```
      node="t0";

      ITSO_GRAM_JOB job2;

      cout << "start app server on " << node << endl;;
      rsl_req = "&(executable=https://";
      rsl_req +=hostname;
      rsl_req += ":20000";
      rsl_req +=get_current_dir_name();
      rsl_req += "/HelloServer) (environment=(LD_LIBRARY_PATH $(HOME)/tmp/lib)
) (count=1) (arguments=";
      char tmpstr[20];
      sprintf(tmpstr,"%ld ",ticket);
      rsl_req += tmpstr;
      rsl_req += node;
      rsl_req += ")";
      // submit it to the GRAM
      cout << rsl_req << endl;
      job2.Submit(node,rsl_req);
      job2.Wait();
```

9.3.4 Compilation

All programs (HelloServer.C, HelloClient.C, and HelloFrontEnd.C) use the class provided by the GNU Common C++ library (thread, sockets management). They are not covered in this publication and are only used to quickly develop the examples. Consequently, the header files located in /usr/local/include/cc++2 and the libraries -lccgnu2 -lpthread -dl must be used during the compilation phase. The library is a prerequisite to run these examples.

1. First, use **`globus-make-header`** to generate a file (globus_header) in which variables are set to appropriate values. globus_header will be included in the Makefile.

   ```
   globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist
   globus_ftp_client globus_ftp_control globus_gram_job globus_common
   globus_gram_client globus_gass_server_ez > globus_header
   ```

2. Use the Makefile shown in Example 9-17 to compile the program.

Example 9-17 Second example Makefile

```
include globus_header

all: HelloServer HelloClient HelloFrontEnd

%.o: %.C
   g++ -c $(GLOBUS_CPPFLAGS) $< -o $@
```

```
HelloServer: HelloServer.C
   g++ -g -I/usr/local/include/cc++2 -L/usr/local/lib -o $@ $^ -lccgnu2
-lpthread -ldl

HelloClient: HelloClient.C
   g++ -g -I/usr/local/include/cc++2 -L/usr/local/lib -o $@ $^ -lccgnu2
-lpthread -ldl

HelloFrontEnd: HelloFrontEnd.o itso_gram_job.o itso_cb.o itso_gass_copy.o
itso_gass_server.o
   g++ -o $@ -I/usr/local/include/cc++2 -L/usr/local/lib $(GLOBUS_CPPFLAGS)
$(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS) -lccgnu2  -lpthread -ldl
```

Then use **`make all`** to obtain the three executables.

9.3.5 Execution

In this publication example, the storage server is m0, which also acts as the front-end server. The source file for GNU Common C++ classes is stored in /tmp/commoncpp2-1.0.8.tar.gz on m0. t0 is the application server. You must generate a valid proxy with **`grid-proxy-init`** before running the test.

On m0, start the Hello front-end server:

```
[globus@m0 globus]$ ./HelloFrontEnd
we are listening on https://m0.itso-maya.com:20000
library compilation on t0
&(executable=https://m0.itso-maya.com:20000/home/globus/JYCode/Compile)
(count=1)
start app server on t0
&(executable=https://m0.itso-maya.com:20000/home/globus/JYCode/HelloServer)
(environment=(LD_LIBRARY_PATH $(HOME)/tmp/lib) ) (count=1)
(arguments=1804289383 t0)
Contact on the server https://t0.itso-tupi.com:57701/3673/1047501872/
Staging file in on: https://t0.itso-tupi.com:57701/3673/1047501872/
Job Finished on: https://t0.itso-tupi.com:57701/3673/1047501872/
binding for: m0.itso-maya.com:4096
```

You can now start the client from another node. You do not need to generate a valid proxy.

```
[john@a0 john]$ ./HelloClient m0
Ticket:1804289383
Hostname:t0
Hello World !
```

On m0, you should see the connection request. The connection is actually handled by the application server t0.

```
accepting from: m0.itso-maya.com:54642
creating session client object
```

9.4 Summary

This chapter has described three different application examples that demonstrate various techniques and concepts of the Globus Toolkit and related capabilities. These samples were purposely kept simple, in that they did not include a lot of error checking or other logic that should be included in a robust business application.

10

Globus Toolkit V3.0

This publication is primarily based on Globus Toolkit V2.2. However, by the time you read this, Globus Toolkit Version 3 will likely be available, at least as beta code.

Though many of the considerations and APIs we have discussed will not change with Globus Toolkit V3, there will be significant changes to the structure of the toolkit and the grid environment itself brought about by the new toolkit and OGSA.

Though many details are not yet available, this chapter is intended to provide a short preview of what might be ahead in Globus Toolkit V3.

© Copyright IBM Corp. 2003. All rights reserved.

10.1 Overview of changes from GT2 to GT3

GT3 is based on OGSI to support the industry standardization of grid protocols. It contains services and features available in GT2, but also allows users to create new services. Some existing features that will be provided include GSI, GRAM, GridFTP, and Information Services. The major difference in the functionality between the two toolkits is that the interface to these features has become an OGSI service. In Globus Toolkit V2, all services were independent of each other; however, in Globus Toolkit V3 all services can be accessed with common interfaces, making it much simpler to develop applications that access these components. Globus Toolkit V3 will also have a more consistent way to obtain information about services, and the need for a GRIS is removed. Each service in GT3 will act as its own GRIS. The data returned from the service will be stored in XML format, whereas in GT2 it is stored in LDIF format. Modification to GT2's MDS service is required to use it with GT3 because of the format difference.

10.1.1 SOAP message security

Grid services must be built on a Grid Security Infrastructure (GSI). Globus Toolkit V2 used the Secure Socket Layer (SSL) protocol for its authentication and message protection. The Globus Toolkit 3 implements a version of the Web Services SecureConversation protocol. This allows for GSI's SSL-based authentication to take place over standard Web Services SOAP messages, which in turn allows for the use of the W3C Web Services Security specifications for message protection: XML-Encryption and XML-Signature.

10.1.2 Creating grid services

One of the main advantages GT3 provides over GT2 is the ability to develop Web services for your grid application. Grid services can be written in C, Java, Python, and so on. However, the client to the grid service must be written in a language that provides bindings to WDSL. GT3 provides Java APIs for programming OGSA services. More information on creating grid services and clients in Java can be found at:

http://www-unix.globus.org/ogsa/docs/alpha3/java_programmers_guide.html

C can be used to write clients to grid services. However, there is currently not an adequate WDSL-to-C compiler. More information on OGSA client-side C implementations can be found at:

http://www-unix.globus.org/ogsa/docs/alpha2/c_users_guide.html

10.1.3 Security - proxies

The new implementation of the GSI libraries (GSI-3) will accept proxy certificates in either GT 2.2 or GT 3.0 proxy certificate formats. The proxy certificate format has been updated to bring it into compliance with the latest proxy certificate draft specification in the Global Grid Forum. This code allows GT 2.2 and 2.4 proxy certificates to be used to authenticate with GT 3.0 services, offering a backwards-compatible migration path from GT 2.2 to GT 3.0.

10.1.4 SOAP GSI plugin for C/C++ Web services

The gSOAP C++ Web services platform is being extended to work with OGSA services and to provide a full interoperability between C/C++ and Java Web services using GSI. A GSI module is used by gSOAP to support the Globus Toolkit 3 security mechanism. This module is tracking with the OGSI evolution and currently supports httpg binding as delegation for job submission.

The gSOAP plugin is available from the following Web site:

http://sara.unile.it/~cafaro/gsi-plugin.html

The gSOAP toolkit provides a SOAP-to-C/C++ language binding for the development of SOAP Web services and clients. It is used in the C implementation of the Globus Toolkit 3 to implement the SOAP protocol. The gSOAP stub and skeleton compiler for C and C++ was developed by Robert van Engelen of Florida State University. See the following Web site for more information.

http://gsoap2.sourceforge.net

10.2 OGSI implementation

Open Grid Service Infrastructure (OGSI) addresses detailed specifications of the interfaces that a service must implement in order to fit into the OGSA framework.

The OGSA architecture and OGSI infrastructure provides a common framework for grid services, so developing new OGSI-compliant services (or specialized versions of existing services) is quite straightforward. Every OGSI-compliant service can be used and managed via common interfaces, so building systems and applications with OGSI-compliant services is much easier.

OGSI software provides mandatory Grid service features, such as service invocation, lifetime management, a service data interface, and security interfaces, that ensure a fundamental level of interoperability among all grid services.

10.3 Open Grid Service Architecture (OSGA)

OGSA draws on the same infrastructure as used in Web services: XML, SOAP, WSDL, and WSIL. However, there are some important conceptual and practical extensions that arise from the need to address a dynamic grid environment providing mechanisms to create and discover customized service instances with controlled, fault-resilient, and secure management of distributed atomic or collective services, often with a long-lived state.

Four important concepts in OGSA are:

- *Naming*: Each grid service instance is globally, uniquely, and for all time named by a Grid Service Handle (GSH).
- *Factories*: Create new grid service instances and maintain a group of service data elements that can be queried. Factories play the role of a gatekeeper or xinet daemons for grid services. A factory will also have an associated registry to keep track of these instances and enable discovery. The OGSA defines registries as places to store various kinds of information about grid resources.
- *Instances*: The GSH is just a minimal name in the form of a URI and does not carry enough information to allow a client to communicate directly with the service instance. Instead, a GSH must be mapped to a Grid Service Reference (GSR) via the registry.
- *Stateful*: A grid service instance has a state. A process can be initiated via a method call on a service port type and its state checked at a later time using the GSR.

A GSR contains all information that a client may require to communicate with the service via one or more protocols. While a GSH is valid for the entire lifetime of the grid service instance, a GSR may become invalid, therefore requiring a client to use the mapping service to acquire a new GSR appropriate to a particular binding using the GSH.

A GSR is encoded using WSDL, so that a WSDL document should be the minimal information required to fully describe how to reach the particular grid service instance.

Globus Toolkit V3 is roughly the Globus Toolkit 2 that uses OGSA architecture to make its components available via Web services.

10.4 Globus grid services

The following sections describe a few of the services that will be available with Globus Toolkit V3. Again, this is early information, and may change before general availability. It is presented here to provide the reader with a flavor of what the services in Globus Toolkit V3 will look like.

10.4.1 Index Services

Index Services is like the MDS feature in GT2. It provides information about the grid services in XML format. Unlike GT2, there is no need for a GRIS because each service has a set of information associated with itself. This information is stored in a standard way, making it easy to retrieve and understand the service data. Each service is required to report common service data and any additional data is optional, allowing users to get a standard set of information from any grid service.

10.4.2 Service data browser

This allows users to view the details of the grid services available and the Web Server Description Language of those services, within a GUI.

10.4.3 GRAM

With the Globus Toolkit V3, users will submit jobs by the way of Web services. The GRAM architecture will be rendered by OGSA via five services:

- The Master Managed Job Factory Service (MMJFS) that is responsible for exposing the virtual GRAM service to the outside world. The Master uses the Service Data Aggregator to collect and populate local Service Data Elements, which represent local scheduler data (freenodes, totalnodes) and general host information (host, cpu type, host OS).
- The Managed Job Factory Service (MJFS) that is responsible for starting a new MJS instance. It exposes only a single Service Data Element, which is an array of Grid Services Handles of all active MJS instances.
- The Managed Job Service (MJS) that is an OGSA service that can submit a job to a local scheduler, monitor its status, and send notifications. The MJS starts two File Streaming Factory Services for stdout and the stderr of the job. Their GSHs are stored in the MFS Service Data Element.

  A Service Data Element (SDE) is an XML element containing information about a service that is identified by name and type, which may contain any XML information and that may be queried or subscribed.

- The File Stream Factory Service that is responsible for creating new instances of a File Stream Service.
- The File Stream Service that is an OGSA service that given a destination URL will a stream from the local file the factory was created to stream (stdout or stderr) to the destination URL.

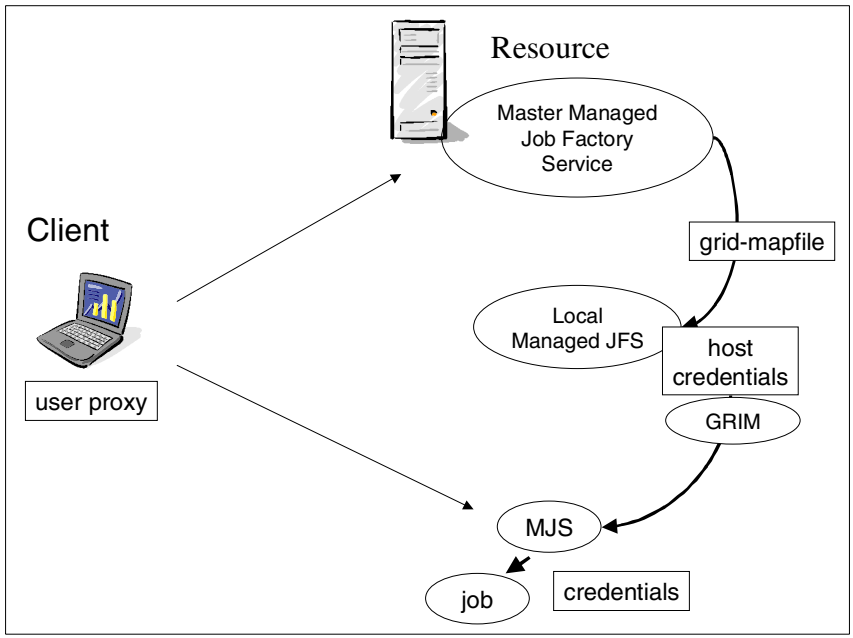


Figure 10-1 Globus Toolkit V3 job invocation

The user then signs this request with her GSI proxy credentials and sends the signed request to the Master Managed Job Factory Service (MMJFS) on the resource that provides a function similar to the Globus Toolkit 2 gatekeeper. The MMJFS still determines the local account in which the job should be run by using a grid-mapfile, as for the Globus Toolkit 2.

Grid Resource Identity Mapper (GRIM) is a setuid program that accesses the local host credentials and from them generates a proxy for the Local Managed Job Factory Service (LMJFS). This proxy credential has embedded in it the user's grid identity, local account name, and local policy about the user.

LMJFS invokes a Managed Job Service (MJS) with the request and returns the address of the MJS to the user. The user then connects to the MJS.

The Globus Toolkit V3 RSL language will be described in XML format even if the functionality remains similar to Globus Toolkit 2. The Managed Job Service will translate it into scheduler-specific language.

Example 10-1 Globus Toolkit V3 RSL example

```
<rsl: rsl <!--- insert GRAM RSL Namespace ---><gram:job>
      <gram:executable>
         <rsl:pathElement path="/bin/ls"/>
      </gram:executable>
      <gram:directory>
         <rsl:pathElement path="/tmp"/>
      </gram:directory>
      <gram:arguments>
         <gram:argument>-l</gram:argument>
         <gram:argument>-a</gram:argument>
      </gram:arguments>
  </gram:job>
</rsl:rsl>
```

Managed Job Factory portType

The Managed Job Factory Service defines an OGSI/WSDL interface for submitting, monitoring, and controlling a job. It is used by a GRAM client to submit a job.

The CreateService operation of the Managed Job Factory portType prepares a job for submission. It takes as input parameter RSL xml document specifying the job to be run and returns the Grid Service Reference (GSR) to MJS as an output parameter that is a WSDL definition of the MJS instance.

The Service Data Element of the Managed Job Factory portType lists the GSHs of MJS instances.

Managed Job portType

The Start operation submits a job.

The MJS will clean up everything when the job is destroyed: It cleans up directories, files, Gass cache, and so on.

The Service Data Elements include:

- Job status
- GSH to File Stream Factory Service for job's stdout
- GSH to File Stream Factory Service for job's stderr

File Stream Factory portType

The CreateService operation prepares to stream a job's stdout or stderr to a destination URL. The input parameter is the destination URL and the output parameter returns the GSH.

The StartStreaming operation starts the streaming to the destination URL.

GRAM Client Interface

Both a C and Java API will be provided in the Globus Toolkit V3 to communicate with the Master Job Service.

An API translator will be also provided in the Globus Toolkit V3 for the Globus C API, the Java and Python Cog Kit, that will translate the Globus Toolkit 2 RSL format into the Globus Toolkit V3 RSL format based on XML.

10.4.4 Reliable File Transfer Service (RFT)

This is an OGSA-based service that provides interfaces for controlling and monitoring third-party file transfers using GridFTP servers. It is similar to the GT2 globus-url-copy tool.

10.4.5 Replica Location Service (RLS)

Large sites often replicate data to provide quick and easy access to data. Distributing replicas of the data reduces data access latency. RLS maintains a registry of information about where replica data resides and makes it easy to find data locations.

10.5 Summary

Fundamentally, the Globus Toolkit V3 will be a Web service-based variation of Globus Toolkit 2. All components like GRAM and GridFTP will remain. But the infrastructure will change to use Web services. Globus Toolkit V3 implements Java wrappers to the Globus Toolkit 2 components and implements new functionality.

RSL and proxy certificate formats are slightly modified to adopt some new standards, but some tools or translators are provided for an easy migration from the Globus Toolkit 2 to the Globus Toolkit V3.

The Globus Toolkit V3 will provide an implementation both in Java and in C, and a client API will be provided in C, Java, and Python.

Applications developed with Globus Toolkit 2 should be easily portable to Globus Toolkit V3.

A

Grid qualification scheme

In this appendix, criteria are presented that may be used as a starting point to determine the suitability of a grid environment for an application.

The criteria are intended to address the two main aspects of architecture, functional and operational, as identified in the article "*A standard for architecture description*" (see bibliography).

Note: Please note that this is not an exhaustive list, but rather presents a summary from the considerations presented in the earlier chapters of this publication. The reader is urged to take into account the specifics of the people, process, and technologies being considered, and to recognize that a full evaluation may require evaluation of a combination of these criteria rather than considering each criteria in isolation.

© Copyright IBM Corp. 2003. All rights reserved.

A suggested grid application qualification scheme

The architecture considerations for a grid application lead to a qualification scheme that highlights the solution requirements and criteria that impact building a grid application.

Table A-1 on page 299 lists several criteria to consider when evaluating an application for grid enabling and deployment.

For each criterion in the table, the following are shown:

Range Provides a summary of the possible values for the criterion.

Importance A weight factor that provides a subjective assessment of how important the criterion is for the application under assessment.

Effort A weight factor that assesses how much effort would be required to modify the application. The scale here (low to high) could identify a range from no effort/simple to a show stopper at the other extreme.

Skills A weight factor of the available skills of the project team.

Resources A weight factor that assesses the capacity and type of resources in the grid (for example, operating system, file system, network, and so on).

Comments Any comments the evaluator wishes to record.

Each criterion would be analyzed and a rating provided for the relevant weight factors, bearing in mind that many of the criteria may have a combined impact on grid enabling the application.

A suggested range of High (H), Medium (M), or Low (L) is shown, but this could obviously be changed to a numeric or other scale.

After evaluating the application against the chosen criteria, show stoppers should be identified and potential workarounds addressed.

As described above, the table identifies several weight factors that could be used to evaluate an application. The factors shown in the table are not exhaustive and the reader may wish to consider adding additional factors (for example, technologies or architecture pattern used) specific to their environments.

Table A-1 Qualification scheme for grid applications

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
1	Job flow	Parallel -> networked -> serial					
2	# different jobs	Single job -> multiple jobs					
3	Sub-jobs depth	No subjobs -> deeply staged subjobs					
4	Job types	Batch -> simple -> parallel application jobs -> EJBs based jobs -> complex jobs					
5	OS depend-ent	Independent -> strongly depending					
6	Memory size needed per job	Small -> large					
7	DLL in place	Standard DLLs -> specific DLLs					
8	Compiler settings	No compiler -> standard settings -> special settings					

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
10	Runtime environ-ment	None required -> standard runtime -> special runtime required					
11	Applica-tion server	None required -> simple beans/JSP -> EJB -> specific needs					
12	Foreign applica-tion	None required -> standard applications -> special settings/installa-tion					
14	Hardware depend-ent	None -> standard IT devices -> special IT devices -> special other devices					
15	Redund-ant job execution	Not required -> heavily depending on					
16	Scaveng-ing grid	All jobs individualized for scavenging -> not suitable for scavenging					

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
17	Job data I/O	Command line parameter -> message queue -> data file -> database -> APIs					
18	Shared data access	RO files -> RO DBMS -> RW files -> RW DBMS					
19	Temp-orary data space	Small -> nearly unlimited (check out concurrent jobs on each node)					
20	Network bandwidth	Small -> high speed network LAN -> WAN					
21	Time-sensitive data	Data always valid -> time depending data values					
22	Data type: Character sets	Commonly available unicode in SBCS network -> different unicode in DBCS -> inconformity of character codes on network					

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
23	Data type: Multi-media formats	Uniform use of set of multimedia formats -> mixed use of formats					
24	Data encryp-tion	Uniform use of encryption techniques available -> varying use of encryption techniques on network					
25	Security policy	Commonly agreed on among all grid users -> discrepancies between involved parties					
26	Time con-straints	No time restrictions apply -> strong need for timely execution and data provisioning					
27	Migration needs	Grid in fixed environment-> grid application based on common standard -> grid likely to migrate on different platforms					

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
28	Data separable per job	Data easily separable -> some solvable data interdependenc-ies -> data inseparable					
29	Amount of data	Small amount of I/O data per job -> large amount of data handled by single jobs					
30	Job topology	Simple job topology (job-node-data) -> complex job topology					
31	Data topology	Simple data topology (data-job-node) -> complex data topology					
32	Network scalability	High upper limit in scalability graph -> low upper limit					
33	Software licensing	All permissive -> all restrictive					

#	Qualification criteria		Weight factors (H-M-L)				Comment show-stoppers -special care -base values
	Item	Range (low to high efforts)	Import-ance	Effort	Skills	Re-sources	
34	Billing service	Not required -> simple direct billing -> complex billing including thrid parties					
35	Single user interface	Not required -> standard UI -> integrated common UI					

B

C/C++ source code for examples

This appendix contains C/C++ source code for various modules that were referenced throughout this publication. It should be noted that this source code is provided as is. Though it was compiled and executed in our specific environment, it has not been thoroughly tested and is meant to provide guidance and examples of how to accomplish certain tasks.

© Copyright IBM Corp. 2003. All rights reserved.

Globus API C++ wrappers

The following examples show some C++ wrappers for some common Globus services.

ITSO_GASS_TRANSFER

This class provides methods to easily transfer a file from one location to another. The GLOBUS_FILE class is use to refer to a locally stored file, and GLOBUS_URL is used to refer to a remotely stored file that can be reached by either http, https, or gsiftp protocol.

The method setURL() of the GLOBUS_URL class is used to define the URL.

The method Transfer() of the ITSO_GASS_TRANSFER class executes the transfer, and the two arguments are respectively the source file and the destination file. The two arguments can either be of GLOBUS_FILE or GLOBUS_URL type.

The Transfer() is non-blocking, so the Wait() method should be called later in the code to wait for the completion of the transfer.

itso_gass_copy.h

```
#ifndef ITSO_GASS_COPY_H
#define ITSO_GASS_COPY_H
#include “globus_common.h”
#include “globus_gass_copy.h”
#include “itso_cb.h”
#include <cstdlib>
#include <cstdio>
#include <iostream>
#include <cstring>
#include <string>


class GLOBUS_FILE {
   globus_io_handle_t *io_handle;
   int       file_fd;
      public:
   GLOBUS_FILE();
   GLOBUS_FILE(char* );
   ~GLOBUS_FILE();
   globus_io_handle_t * GLOBUS_FILE::get_globus_io_handle();
};

class GLOBUS_URL {
```

```
   globus_url_t url;
   globus_gass_copy_url_mode_t url_mode;
   char*        URL;
   public:
      GLOBUS_URL();
      ~GLOBUS_URL();
      bool setURL(char* destURL);
      bool setURL(string destURL);
      globus_gass_copy_url_mode_t getMode();
      char* getScheme();
      char* getURL();
};

class ITSO_GASS_TRANSFER_EXCEPTION { };

class ITSO_GASS_TRANSFER : public ITSO_CB {
   globus_gass_copy_handle_t          gass_copy_handle;
   globus_gass_copy_handleattr_t      gass_copy_handleattr;
   globus_gass_transfer_requestattr_t*dest_gass_attr;
   globus_gass_copy_attr_t dest_gass_copy_attr;
   globus_gass_transfer_requestattr_t*source_gass_attr;
   globus_gass_copy_attr_t source_gass_copy_attr;
   globus_gass_copy_url_mode_t source_url_mode;
   globus_gass_copy_url_mode_t dest_url_mode;
   globus_ftp_client_operationattr_t*dest_ftp_attr;
   globus_ftp_client_operationattr_t*source_ftp_attr;
   void setSource(GLOBUS_URL& );
   void setDestination(GLOBUS_URL& );
   public:
   ITSO_GASS_TRANSFER();
   ~ITSO_GASS_TRANSFER();
   void Transfer(GLOBUS_FILE& , GLOBUS_URL& );
   void Transfer(GLOBUS_URL&,GLOBUS_FILE&);
   void Transfer(GLOBUS_URL& ,GLOBUS_URL& );
};

#endif
```

itso_gass_copy.C

```
/*******************************************
 *  For a more complete example
 *  see globus-url-copy.c
 *******************************************/
#include “itso_gass_copy.h”

GLOBUS_FILE::GLOBUS_FILE() {};
GLOBUS_FILE::GLOBUS_FILE(char* filename) {
```

```
   io_handle =(globus_io_handle_t *)
globus_libc_malloc(sizeof(globus_io_handle_t));
   file_fd=open(filename,O_RDONLY);
       /* convert file  into a globus_io_handle */
       globus_io_file_posix_convert(file_fd,
                                    GLOBUS_NULL,
                                   io_handle);
};

GLOBUS_FILE::~GLOBUS_FILE(){
   close(file_fd);
   globus_libc_free(io_handle);
};

globus_io_handle_t * GLOBUS_FILE::get_globus_io_handle() {
   return io_handle;
};

GLOBUS_URL::GLOBUS_URL() {};
GLOBUS_URL::~GLOBUS_URL() {
   free(URL);
};
bool GLOBUS_URL::setURL(char* destURL) {
    //check if this is a valid URL
   if (globus_url_parse(destURL, &url) != GLOBUS_SUCCESS) {
             cerr << “can not parse destURL” << destURL << endl;
        return false;
   }
   //determine the transfer mode
   if (globus_gass_copy_get_url_mode(destURL, &url_mode) != GLOBUS_SUCCESS) {
        cerr << “failed to determine mode fmeor destURL” << destURL << endl;
        return false;
   };
   URL=strdup(destURL);
   return true;
};

bool GLOBUS_URL::setURL(string url) {
   return setURL(const_cast<char*>(url.c_str()));
}

globus_gass_copy_url_mode_t GLOBUS_URL::getMode() {
   return url_mode;
};

char* GLOBUS_URL::getScheme() {
   return url.scheme;
}
```

```
char* GLOBUS_URL::getURL() {
   return URL;
}

//************************************************************
//
//************************************************************
namespace itso_gass_copy {
static void
url_copy_callback(
          void * callback_arg,
          globus_gass_copy_handle_t * handle,
          globus_object_t * error)
{
            globus_bool_t         use_err = GLOBUS_FALSE;
       ITSO_CB* monitor = (ITSO_CB*)  callback_arg;

            if (error != GLOBUS_SUCCESS)
       {
            cerr << " url copy error:" <<
globus_object_printable_to_string(error) << endl;
            //monitor->setError(error);
          }
       monitor->setDone();
        return;
};
}

ITSO_GASS_TRANSFER::ITSO_GASS_TRANSFER() {
   // handlers initialisation
   // first the attributes
   // then the handler
   globus_gass_copy_handleattr_init(&gass_copy_handleattr);
   globus_gass_copy_handle_init(&gass_copy_handle, &gass_copy_handleattr);
};

ITSO_GASS_TRANSFER::~ITSO_GASS_TRANSFER() {
   globus_gass_copy_handle_destroy(&gass_copy_handle);
   if (source_url_mode == GLOBUS_GASS_COPY_URL_MODE_FTP)
      globus_libc_free(source_ftp_attr);
   if (dest_url_mode == GLOBUS_GASS_COPY_URL_MODE_FTP)
      globus_libc_free(dest_ftp_attr);
   if (source_url_mode == GLOBUS_GASS_COPY_URL_MODE_GASS)
      globus_libc_free(source_gass_attr);
   if (dest_url_mode == GLOBUS_GASS_COPY_URL_MODE_GASS)
      globus_libc_free(dest_gass_attr);
}

void ITSO_GASS_TRANSFER::setSource(GLOBUS_URL& source_url) {
```

```
   globus_gass_copy_attr_init(&source_gass_copy_attr);
   source_url_mode=source_url.getMode();
   if (source_url_mode == GLOBUS_GASS_COPY_URL_MODE_FTP) {
      source_ftp_attr = (globus_ftp_client_operationattr_t*)
globus_libc_malloc (sizeof(globus_ftp_client_operationattr_t));

      globus_ftp_client_operationattr_init(source_ftp_attr);
      globus_gass_copy_attr_set_ftp(&source_gass_copy_attr,
                  source_ftp_attr);
   }
   else if (source_url_mode == GLOBUS_GASS_COPY_URL_MODE_GASS) {
      source_gass_attr = (globus_gass_transfer_requestattr_t*)
globus_libc_malloc (sizeof(globus_gass_transfer_requestattr_t));

globus_gass_transfer_requestattr_init(source_gass_attr,source_url.getScheme());
      globus_gass_copy_attr_set_gass(&source_gass_copy_attr,
source_gass_attr);
      globus_gass_transfer_requestattr_set_file_mode(
            source_gass_attr,
            GLOBUS_GASS_TRANSFER_FILE_MODE_BINARY);
      globus_gass_copy_attr_set_gass(&source_gass_copy_attr,
                                 source_gass_attr);
   };
};


void ITSO_GASS_TRANSFER::setDestination(GLOBUS_URL& dest_url) {
   globus_gass_copy_attr_init(&dest_gass_copy_attr);
   dest_url_mode=dest_url.getMode();
   if (dest_url_mode == GLOBUS_GASS_COPY_URL_MODE_FTP) {
      dest_ftp_attr = (globus_ftp_client_operationattr_t*)globus_libc_malloc
(sizeof(globus_ftp_client_operationattr_t));
      globus_ftp_client_operationattr_init(dest_ftp_attr);
      globus_gass_copy_attr_set_ftp(&dest_gass_copy_attr,
                  dest_ftp_attr);
   }
   else if (dest_url_mode == GLOBUS_GASS_COPY_URL_MODE_GASS) {
      dest_gass_attr = (globus_gass_transfer_requestattr_t*)globus_libc_malloc
(sizeof(globus_gass_transfer_requestattr_t));
       globus_gass_transfer_requestattr_init(dest_gass_attr,
dest_url.getScheme());
      globus_gass_copy_attr_set_gass(&dest_gass_copy_attr, dest_gass_attr);
      globus_gass_transfer_requestattr_set_file_mode(
            dest_gass_attr,
            GLOBUS_GASS_TRANSFER_FILE_MODE_BINARY);
      globus_gass_copy_attr_set_gass(&dest_gass_copy_attr,
                                 dest_gass_attr);
   };
```

```
};

void ITSO_GASS_TRANSFER::Transfer(GLOBUS_FILE& globus_source_file, GLOBUS_URL&
destURL) {
       setDestination(destURL);
globus_result_t result = globus_gass_copy_register_handle_to_url(
         &gass_copy_handle,
                     globus_source_file.get_globus_io_handle(),
              destURL.getURL(),
             &dest_gass_copy_attr,
         itso_gass_copy::url_copy_callback,
                    (void *) this );
};

void ITSO_GASS_TRANSFER::Transfer(GLOBUS_URL& sourceURL,GLOBUS_FILE&
globus_dest_file) {
       setSource(sourceURL);
globus_result_t result = globus_gass_copy_register_url_to_handle(
         &gass_copy_handle,
         sourceURL.getURL(),
             &source_gass_copy_attr,
                     globus_dest_file.get_globus_io_handle(),
         itso_gass_copy::url_copy_callback,
                    (void *) this );
};

void ITSO_GASS_TRANSFER::Transfer(GLOBUS_URL& sourceURL,GLOBUS_URL& destURL) {
       setSource(destURL);
       setDestination(destURL);
globus_result_t result = globus_gass_copy_register_url_to_url(
                    &gass_copy_handle,
                       sourceURL.getURL(),
               &source_gass_copy_attr,
                       destURL.getURL(),
                       &dest_gass_copy_attr,
         itso_gass_copy::url_copy_callback,
                       (void *) this);
};
```

ITSO_GLOBUS_FTP_CLIENT

A wrapper for GridFTP capabilities.

itso_globus_ftp_client.h

```
#ifndef ITSO_ITSO_GLOBUS_FTP_CLIENT_H
#define ITSO_ITSO_GLOBUS_FTP_CLIENT_H
#include <cstdio>
```

```
#include <iostream>
#include “globus_ftp_client.h”
#include “itso_cb.h”

#define _(a) r=a;\
        if (r!=GLOBUS_SUCCESS) {\
       cerr << globus_object_printable_to_string(globus_error_get(r));\
       exit(1);\
        }


#define MAX_BUFFER_SIZE 2048
#define SUCCESS 0

//***************************************************************************


class ITSO_GLOBUS_FTP_CLIENT : public ITSO_CB {
   FILE*     fd;
    globus_byte_t           buffer[MAX_BUFFER_SIZE];
   globus_ftp_client_handle_t              handle;
   public:
    ITSO_GLOBUS_FTP_CLIENT(char*,char*);
   ~ITSO_GLOBUS_FTP_CLIENT();
   void StartTransfer();
   void Transfer( globus_byte_t*, globus_size_t&,globus_off_t&);
};
#endif
#ifndef ITSO_ITSO_GLOBUS_FTP_CLIENT_H
#define ITSO_ITSO_GLOBUS_FTP_CLIENT_H
#include <cstdio>
#include <iostream>
#include “globus_ftp_client.h”
#include “itso_cb.h”

#define _(a) r=a;\
        if (r!=GLOBUS_SUCCESS) {\
       cerr << globus_object_printable_to_string(globus_error_get(r));\
       exit(1);\
        }

#define MAX_BUFFER_SIZE 2048
#define SUCCESS 0

//***************************************************************************
class ITSO_GLOBUS_FTP_CLIENT : public ITSO_CB {
   FILE*     fd;
    globus_byte_t           buffer[MAX_BUFFER_SIZE];
   globus_ftp_client_handle_t              handle;
```

```
   public:
    ITSO_GLOBUS_FTP_CLIENT(char*,char*);
   ~ITSO_GLOBUS_FTP_CLIENT();
   void StartTransfer();
   void Transfer( globus_byte_t*, globus_size_t&,globus_off_t&);
};
#endif
```

itso_globus_ftp_client.C

```
#include “itso_globus_ftp_client.h”

namespace itso_globus_ftp_client {
static
void
done_cb(
        void*                user_arg,
        globus_ftp_client_handle_t *            handle,
        globus_object_t *                       err)
{
    ITSO_CB* f=(ITSO_CB*) user_arg;
if(err)
    {
        cerr << globus_object_printable_to_string(err);
    };
    f->setDone();
    return;
};

static
void
data_cb(
    void *                                      user_arg,
    globus_ftp_client_handle_t *                handle,
    globus_object_t *                           err,
    globus_byte_t *                             buffer,
    globus_size_t                               length,
    globus_off_t                                offset,
    globus_bool_t                               eof)
{
    ITSO_GLOBUS_FTP_CLIENT* l = (ITSO_GLOBUS_FTP_CLIENT*) user_arg;

    if(err)
    {
        fprintf(stderr, “%s”, globus_object_printable_to_string(err));
    }
    else
    {
        if(!eof)
```

```
      l->Transfer(
                                buffer,
                                length,
                                offset
      );
    } /* else */
    return;
} /* data_cb */
};

ITSO_GLOBUS_FTP_CLIENT::ITSO_GLOBUS_FTP_CLIENT(char* f,char* dst) {
      fd = fopen(f,"r");
      globus_ftp_client_handle_init(&handle,  GLOBUS_NULL);
      globus_result_t r;
      globus_ftp_client_put(
                             &handle,
                             dst,
                             GLOBUS_NULL,
                             GLOBUS_NULL,
                             itso_globus_ftp_client::done_cb,
                             this);
   };
ITSO_GLOBUS_FTP_CLIENT::~ITSO_GLOBUS_FTP_CLIENT() {
      fclose(fd);
      globus_ftp_client_handle_destroy(&handle);
   };
void ITSO_GLOBUS_FTP_CLIENT::StartTransfer() {
        int rc;
           rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
           globus_ftp_client_register_write(
            &handle,
               buffer,
            rc,
               0,
            feof(fd) != SUCCESS,
          itso_globus_ftp_client::data_cb,
               (void*) this);
   };

void ITSO_GLOBUS_FTP_CLIENT::Transfer(
       globus_byte_t*                              buffer,
       globus_size_t&                                length,
       globus_off_t&                                 offset
           )
{
   int rc;
        rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
        if (ferror(fd) != SUCCESS)
        {
```

```
            printf("Read error in function data_cb; errno = %d\n", errno);
            return;
        }
        globus_ftp_client_register_write(
           &handle,
           buffer,
           rc,
           offset + length,
           feof(fd) != SUCCESS,
      itso_globus_ftp_client::data_cb,
           (void *) this);
};
```

ITSO_CB

Below is a sample callback mechanism.

itso_cb.h

```
#ifndef ITSO_CB_H
#define ITSO_CB_H
#include <cstdio>
#include <iostream>
#include <globus_common.h>

class ITSO_CB {
   globus_mutex_t mutex;
   globus_cond_t cond;
   globus_bool_t done;
     public:
   ITSO_CB() {
      globus_mutex_init(&mutex, GLOBUS_NULL);
      globus_cond_init(&cond, GLOBUS_NULL);
      done = GLOBUS_FALSE ;
   };
   ~ITSO_CB() {
              globus_mutex_destroy(&mutex);
              globus_cond_destroy(&cond);
   };
   globus_bool_t IsDone();
        void setDone();
        void Continue();
   virtual void Wait();
};
#endif
```

itso_cb.C

```
#include "itso_cb.h"
```

```
globus_bool_t ITSO_CB::IsDone() { return done; };

void ITSO_CB::setDone() {
              globus_mutex_lock(&mutex);
              done = GLOBUS_TRUE;
         globus_cond_signal(&cond);
         globus_mutex_unlock(&mutex);
            }

void ITSO_CB::Continue() {
              globus_mutex_lock(&mutex);
              done = GLOBUS_FALSE;
         globus_mutex_unlock(&mutex);
            }

void ITSO_CB::Wait() {
   globus_mutex_lock(&mutex);
    while(!IsDone())
        globus_cond_wait(&cond, &mutex);
   globus_mutex_unlock(&mutex);
};
```

ITSO_GRAM_JOB

This class provides methods to easily submit a job to a Globus grid. It works in an asynchronous way:

- The Submit() method takes a host name and the RSL string to submit the job and returns immediately.
- The Wait() method waits for the completion of the job.

The class is derived from ITSO_CB and provides the following methods to check the status of the job:

- IsDone() to check the status of the job (finished or not).
- HasFailed() to check if the GRAM submission has failed. Note that this method will not detect if the executable aborted during execution or if it hangs.

The class can be used several times to submit different jobs with either a different host name or a different RSL string. Use the Continue() method to be able to use the Submit() method again.

itso_gram_job.h

```
#ifndef ITSO_GRAM_JOB_H
#define ITSO_GRAM_JOB_H
```

```
#include <cstdio>
#include <string>
#include "globus_gram_client.h"
#include "itso_cb.h"

class ITSO_GRAM_JOB : public ITSO_CB {
   char* job_contact;
        char* callback_contact; /* This is the identifier for
                                 * globus_gram_job_request

              */
   bool   failed;// used to check if a job has failed
     public:
   ITSO_GRAM_JOB();
   ~ITSO_GRAM_JOB();
   bool Submit(string,string);
   void Cancel();
   void SetJobContact(const char*);
   void Wait();
   void SetFailed();
   bool HasFailed();
};
#endif
```

itso_gram_jobs_callback.h

```
#ifndef ITSO_GRAM_JOBS_CALLBACK_H
#define ITSO_GRAM_JOBS_CALLBACK_H
#include <cstdio>
#include <string>
#include <map>
#include "globus_gram_client.h"
#include "itso_cb.h"

class ITSO_GRAM_JOBS_CALLBACK;
class ITSO_GRAM_JOB;

class ITSO_GRAM_JOBS_CALLBACK {
   globus_mutex_t JobsTableMutex;
   char*  callback_contact; /* This is the identifier for
                                 * the callback, returned by
                                 * globus_gram_job_request
                  */
   map<string,ITSO_GRAM_JOB*> JobsTable;
   void   Lock();
   void   UnLock();
     public:
   ITSO_GRAM_JOBS_CALLBACK();
   ~ITSO_GRAM_JOBS_CALLBACK();
```

```
   void      Add(string,ITSO_GRAM_JOB*);
   void      Remove(char*);
   char*     GetURL();
   ITSO_GRAM_JOB*GetJob(char*);
};

class ITSO_GRAM_JOB : public ITSO_CB {
   char*        jobcontact;
   bool         failed;
   ITSO_GRAM_JOBS_CALLBACK* callback;
      public:
   ITSO_GRAM_JOB(ITSO_GRAM_JOBS_CALLBACK* f) :
failed(false),jobcontact(NULL),callback(f) {};
   ~ITSO_GRAM_JOB() {};
   bool Submit(string,string);
   void Cancel();
   void SetJobContact(char*);
   void Wait();
   void SetFailed();
   bool HasFailed();
};
#endif
```

itso_gram_jobs_callback.C

```
#include "itso_gram_jobs_callback.h"

namespace itso_gram_jobs {
static void callback_func(void * user_callback_arg,
                  char * job_contact,
                  int state,
                  int errorcode)
{
    ITSO_GRAM_JOBS_CALLBACK*  Monitor = (ITSO_GRAM_JOBS_CALLBACK*)
user_callback_arg;

    ITSO_GRAM_JOB* job = Monitor->GetJob(job_contact);

    switch(state)
    {
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
   cout << "Staging file in on: " << job_contact << endl;
   break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
   cout << "Staging file out  on: " << job_contact << endl;
   break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
   break; /* Reports state change to the user */
```

```
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
   break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
   Monitor->Remove(job_contact);
   job->SetFailed();
   job->setDone();
   cerr << "Job Failed on: " << job_contact << endl;
   break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
   cout << "Job Finished on: " << job_contact << endl;
   Monitor->Remove(job_contact);
   job->setDone();
   break; /* Reports state change to the user */
    }
}
static void request_callback(void * user_callback_arg,
                             globus_gram_protocol_error_t failure_code,
                             const char * job_contact,
                             globus_gram_protocol_job_state_t state,
                             globus_gram_protocol_error_t errorcode) {
    ITSO_GRAM_JOB*  Request = (ITSO_GRAM_JOB*) user_callback_arg;
    cout << "Contact on the server " << job_contact << endl;

    if (failure_code==0) {
    Request->SetJobContact(const_cast<char*>(job_contact));
    }
    else {
       cout << "Error during the code submission" << endl << "Error Code:" <<
failure_code << endl;
   Request->setDone();
   Request->SetFailed();
    }
}
}

void ITSO_GRAM_JOB::SetJobContact(char* c) {
    jobcontact=c;
    callback->Add(c,this);
};

bool ITSO_GRAM_JOB::Submit(string res, string rsl) {
    failed=false;
    int rc = globus_gram_client_register_job_request(res.c_str(),
                        rsl.c_str(),
                    GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
          callback->GetURL(),
                        GLOBUS_GRAM_CLIENT_NO_ATTR,
```

```
            itso_gram_jobs::request_callback,
            (void*) this);
    if (rc != 0) /* if there is an error */
    {
        printf("TEST: gram error: %d - %s\n",
                rc,
                /* translate the error into english */
                globus_gram_client_error_string(rc));
        return true;
    }
    else {

   return false;
    };
};

void ITSO_GRAM_JOB::Wait() {
   ITSO_CB::Wait();
        /* Free up the resources of the job_contact, as the job is over, and
         * the contact is now useless.
         */
   if (jobcontact!=NULL) {
           globus_gram_client_job_contact_free(jobcontact);
      jobcontact=NULL;
   };
   Continue();
};

void ITSO_GRAM_JOB::Cancel() {
    int rc;
    printf("\tTEST: sending cancel to job manager...\n");

    if ((rc = globus_gram_client_job_cancel(jobcontact)) != 0)
    {
       printf("\tTEST: Failed to cancel job.\n");
       printf("\tTEST: gram error: %d - %s\n",
               rc,
               globus_gram_client_error_string(rc));
    }
    else
    {
       printf("\tTEST: job cancel was successful.\n");
    }
};

void ITSO_GRAM_JOB::SetFailed() {
   failed=true;
};
```

```
bool ITSO_GRAM_JOB::HasFailed() {
   return failed;
};

ITSO_GRAM_JOBS_CALLBACK::ITSO_GRAM_JOBS_CALLBACK() {
   globus_mutex_init(&JobsTableMutex, ITSO_NULL);
    globus_gram_client_callback_allow(
       itso_gram_jobs::callback_func,
                (void *) this,
                 &callback_contact);
   cout << "Gram contact " << callback_contact << endl;
};

char*  ITSO_GRAM_JOBS_CALLBACK::GetURL() {
   return callback_contact;
}

ITSO_GRAM_JOB*  ITSO_GRAM_JOBS_CALLBACK::GetJob(char* s) {
   return JobsTable[s];
}

ITSO_GRAM_JOBS_CALLBACK::~ITSO_GRAM_JOBS_CALLBACK() {
   cout << callback_contact << " destroyed" << endl;
   globus_gram_client_callback_disallow(callback_contact);
   globus_free(callback_contact);
   globus_mutex_destroy(&JobsTableMutex);
};

void ITSO_GRAM_JOBS_CALLBACK::Add(string jobcontact,ITSO_GRAM_JOB* job) {
   Lock();
   JobsTable[jobcontact]=job;
   UnLock();
};

void ITSO_GRAM_JOBS_CALLBACK::Remove(char* jobcontact){
   Lock();
   JobsTable.erase(jobcontact);
   UnLock();
};

void ITSO_GRAM_JOBS_CALLBACK::Lock() { globus_mutex_lock(&JobsTableMutex); };
void ITSO_GRAM_JOBS_CALLBACK::UnLock() { globus_mutex_unlock(&JobsTableMutex);
};
```

itso_gram_job.C

```
#include “itso_gram_job.h”

namespace itso_gram_job {
```

```
static void callback_func(void * user_callback_arg,
                  char * job_contact,
                  int state,
                  int errorcode)
{
    ITSO_GRAM_JOB*  Monitor = (ITSO_GRAM_JOB*) user_callback_arg;

    switch(state)
    {
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN:
          cout << "Staging file in on: " << job_contact << endl;
          break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_OUT:
          cout << "Staging file out  on: " << job_contact << endl;
          break;
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
          cerr << "Job Failed on: " << job_contact << endl;
          Monitor->SetFailed();
          Monitor->setDone();
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
          cout << "Job Finished on: " << job_contact << endl;
          Monitor->setDone();
          break; /* Reports state change to the user */
    }
}

static void request_callback(void * user_callback_arg,
       globus_gram_protocol_error_t       failure_code,
       const char *                       job_contact,
       globus_gram_protocol_job_state_t   state,
       globus_gram_protocol_error_t errorcode)
{
   ITO_GRAM_JOB*  Request = (ITSO_GRAM_JOB*) user_callback_arg;
   cout << "Contact on the server " << job_contact << endl;
    Request->SetRequestDone(job_contact);
}
}

ITSO_GRAM_JOB::ITSO_GRAM_JOB() {
};
```

```
ITSO_GRAM_JOB::~ITSO_GRAM_JOB() {
};

void ITSO_GRAM_JOB::SetRequestDone( const char* j) {
   job_contact = const_cast<char*>(j);
   request_cb.setDone();
}

void ITSO_GRAM_JOB::Submit(string res, string rsl) {
    failed=false;
    globus_gram_client_callback_allow(itso_gram_job::callback_func,
                      (void *) this,
                      &callback_contact);
    int rc = globus_gram_client_register_job_request(res.c_str(),
                         rsl.c_str(),
                         GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL,
                         callback_contact,
                         GLOBUS_GRAM_CLIENT_NO_ATTR,
                         itso_gram_job::request_callback,
                         (void*) this);
    if (rc != 0) /* if there is an error */
    {
        printf("TEST: gram error: %d - %s\n",
                rc,
                /* translate the error into english */
                globus_gram_client_error_string(rc));
        return;
    }
};

void ITSO_GRAM_JOB::Wait() {
   request_cb.Wait();
   ITSO_CB::Wait();
        /* Free up the resources of the job_contact, as the job is over, and
         * the contact is now useless.
         */
        globus_gram_client_job_contact_free(job_contact);
   request_cb.Continue();
   ITSO_CB::Continue();
};

void ITSO_GRAM_JOB::Cancel() {
    int rc;
    printf("\tTEST: sending cancel to job manager...\n");

    if ((rc = globus_gram_client_job_cancel(job_contact)) != 0)
    {
       printf("\tTEST: Failed to cancel job.\n");
       printf("\tTEST: gram error: %d - %s\n",
```

```
                rc,
                globus_gram_client_error_string(rc));
    }
    else
    {
       printf("\tTEST: job cancel was successful.\n");
    }
};

void ITSO_GRAM_JOB::SetFailed() {
   failed=true;
}

bool ITSO_GRAM_JOB::HasFailed() {
   return failed;
}
```

StartGASSServer() and StopGASSServer()

These two functions provide an easy way to start and stop a local GASS server. The StartGASSServer takes one argument (the port number on which the GASS server must listen on).

As the callback function used for globus_gass_server_ez_init() does not take any argument, an object cannot be passed to the callbacks. Consequently, if the application needs to start two local GASS servers, two different callback functions must be used and globus_gass_server_ez_init() must be called twice with a different callback each time.

itso_gass_server.h

```
#ifndef ITSO_GASS_SERVER_H
#define ITSO_GASS_SERVER_H

#include <unistd.h>
#include "globus_common.h"
#include "globus_gass_server_ez.h"
#include <iostream>

namespace itso_gass_server {

void StartGASSServer(int);

void StopGASSServer();

};

#endif
```

itso_gass_server.C

```
#include “itso_gass_server.h”

namespace itso_gass_server {

globus_mutex_t                mutex;
globus_cond_t                 cond;
bool                       done;
globus_gass_transfer_listener_t GassServerListener;

void callback_c_function() {
       globus_mutex_lock(&mutex);
       done = true;
       globus_cond_signal(&cond);
}

void StartGASSServer(int port=10000) {
   // Never forget to activate GLOBUS module
   globus_module_activate(GLOBUS_GASS_SERVER_EZ_MODULE);

   // let s define options for our GASS server
   unsigned long server_ez_opts=0UL;

   //Files openfor writing will be written a line at a time
   //so multiple writers can access them safely.
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_LINE_BUFFER;

   //URLs that have ~ character, will be expanded to the home
   //directory of the user who is running the server
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_EXPAND;

   //URLs that have ~user character, will be expanded to the home
   //directory of the user on the server machine
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_USER_EXPAND;

   //”get” requests will be fullfilled
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_READ_ENABLE;

   //”put” requests will be fullfilled
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_WRITE_ENABLE;

   // for put requets on /dev/stdout will be redirected to the standard
   // output stream of the gass server
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDOUT_ENABLE;

   // for put requets on /dev/stderr will be redirected to the standard
   // output stream of the gass server
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDERR_ENABLE;
```

```
   // “put requests” to the URL https://host/dev/globus_gass_client_shutdown
   // will cause the callback function to be called.  this allows
   // the GASS client to communicate shutdown requests to the server
   server_ez_opts |= GLOBUS_GASS_SERVER_EZ_CLIENT_SHUTDOWN_ENABLE;

   // Secure
   char* scheme=”https”;
   //unsecure
   //char* scheme=”http”;
        globus_gass_transfer_listenerattr_t  attr;
        globus_gass_transfer_listenerattr_init(&attr, scheme);

        //we want to listen on post 10000
   globus_gass_transfer_listenerattr_set_port(&attr, port);

   //Now, we can start this gass server !
        globus_gass_transfer_requestattr_t  * reqattr      = GLOBUS_NULL;
//purpose unknown

        globus_mutex_init(&mutex, GLOBUS_NULL);
        globus_cond_init(&cond, GLOBUS_NULL);
        done = false;

   int err = globus_gass_server_ez_init(&GassServerListener,
                                        &attr,
                                        scheme,
                                        GLOBUS_NULL, //purpose unknown
                                        server_ez_opts,
                callback_c_function); //or GLOBUS_NULL otherwise
                //GLOBUS_NULL); //or GLOBUS_NULL otherwise

   if((err != GLOBUS_SUCCESS)) {
           cerr << “Error: initializing GASS (“ << err << “)” << endl;
          exit(1);
   }

        char *
gass_server_url=globus_gass_transfer_listener_get_base_url(GassServerListener);
   cout << “we are listening on “ << gass_server_url << endl;

};

void StopGASSServer() {
   globus_gass_server_ez_shutdown(GassServerListener);
   globus_module_deactivate(GLOBUS_GASS_SERVER_EZ_MODULE);
};

};
```

ITSO broker

This is a simple implementation of a broker via the GetLinuxNodes() function. The function takes an integer as the number of required nodes and returns a vector of strings containing the host names.

The broker checks the status of the nodes by using the Globus ping function provided by the globus GRAM module API. Consequently, the function activates and deactivates the Globus GRAM module.

The algorithm takes into account the CPU speed, the number of processors, and the CPU workload, and returns the best nodes available. Only Linux nodes are returned.

Broker.h

```
#ifndef ITSO_BROKER_H
#define ITSO_BROKER_H
/* gris_search.c  */
#include “globus_common.h”
/* LDAP stuff */
#include “lber.h”
#include “ldap.h”
#include <string>
#include <vector>
#include <list>
#include <algorithm>

/* note this should be the GIIS server, but it could be the
   GRIS server if you are only talking to a local machine
   remember the port numbers are different */

#define GRID_INFO_HOST “m0”
#define GRID_INFO_PORT “2135”
#define GRID_INFO_BASEDN “mds-vo-name=maya, o=grid”

namespace itso_broker {


void GetLinuxNodes(vector<string>& res,int n);
}
#endif
```

Broker.C

```
/* gris_search.c  */
#include “globus_common.h”
#include “globus_gram_client.h”
/* LDAP stuff */
```

```
#include “lber.h”
#include “ldap.h”
#include <string>
#include <vector>
#include <algorithm>

/* note this should be the GIIS server, but it could be the
   GRIS server if you are only talking to a local machine
   remember the port numbers are different */

#define GRID_INFO_HOST “m0”
#define GRID_INFO_PORT “2135”
#define GRID_INFO_BASEDN “mds-vo-name=maya, o=grid”

//*********************************************************************
// This is a basic implementation of a broker.  It checks all available
// Linux nodes and teir CPU usage.  Use GetLinuxNodes().  The first
// parameter is a vector of strings that will contain the list of ost
// in returns.  The second parameter is the number of nodes requested.
//
// The most interesting part are the ldap calls and the way to proceed
// to retrieve information from the MDS server using OpenLdap C API.
//
// other interesting implementation can couple LDAP information with
// other info in a DB for example that Time zone of the execution host,
// location, service level aggrement with the requester and the
// resources provider ...
//*********************************************************************

namespace itso_broker {

class Host {
   string  hostname;
   longcpu;
   public:
   Host(string h,int c) : hostname(h), cpu(c) {};
   ~Host() { };
   string getHostname() { return hostname; };
   int getCpu() { return cpu; };
};

bool predica(Host* a, Host* b) {
 return (a->getCpu() > b->getCpu());
}

void GetLinuxNodes(vector<string>& res,int n)
{
    LDAP *           ldap_server;
    LDAPMessage *    reply;
```

```
    LDAPMessage *    entry;
    char *           attrs[1];
    char *           server   = GRID_INFO_HOST;
    int              port     = atoi(GRID_INFO_PORT);
    char *           base_dn  = GRID_INFO_BASEDN;

    /* list of attributes that we want included in the search result */
    attrs[0] = GLOBUS_NULL;

    globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE);
    /* Open connection to LDAP server */
    if ((ldap_server = ldap_open(server, port)) == GLOBUS_NULL)
    {
        ldap_perror(ldap_server, "ldap_open");
        exit(1);
    }

    /* Bind to LDAP server */
    if (ldap_simple_bind_s(ldap_server, "", "") != LDAP_SUCCESS)
    {
        ldap_perror(ldap_server, "ldap_simple_bind_s");
        ldap_unbind(ldap_server);
        exit(1);
    }

    /* do the search to find all the Linux available nodes*/
    //string filter= "(objectClass=MdsComputer)(Mds-Os-name=Linux)";
    string filter= "(&(Mds-Os-name=Linux)(Mds-Host-hn=*))";

    if (ldap_search_s(ldap_server, base_dn,
                      LDAP_SCOPE_SUBTREE,
                      const_cast<char*>(filter.c_str()), attrs, 0,
                      &reply) != LDAP_SUCCESS)
    {
        ldap_perror(ldap_server, "ldap_search");
        ldap_unbind(ldap_server);
        exit(1);
    }
    vector<Host*> nodes;

    /* go through the entries returned by the LDAP server. for each
      entry, we must search for the right attribute and then get the
      value associated with it */
    for (entry = ldap_first_entry(ldap_server, reply);
          entry != GLOBUS_NULL;
          entry = ldap_next_entry(ldap_server, entry) )
    {

// cout << endl << ldap_get_dn( ldap_server, entry ) << endl;
```

```
        BerElement * ber;
        char**  values;
        char *  attr;
        char *  answer = GLOBUS_NULL;
    string hostname;
    int cpu;
    int cpu_nb;
    long speed;

        for (attr = ldap_first_attribute(ldap_server,entry,&ber);
             attr != NULL;
             attr = ldap_next_attribute(ldap_server,entry,ber) )
    {
        values = ldap_get_values(ldap_server, entry, attr);
            answer = strdup(values[0]);
            ldap_value_free(values);
       if (strcmp("Mds-Host-hn",attr)==0)
         hostname=answer;
       if (strcmp("Mds-Cpu-Free-15minX100",attr)==0)
          cpu=atoi(answer);
       if (strcmp("Mds-Cpu-Total-count",attr)==0)
          cpu_nb=atoi(answer);
       if (strcmp("Mds-Cpu-speedMHz",attr)==0)
          speed=atoi(answer);
      //     printf("%s %s\n", attr, answer);

    }
    // check if we can really use this node
    if (!globus_gram_client_ping(hostname.c_str()))
       nodes.push_back(new Host(hostname,speed*cpu_nb*cpu/100));

     };
     sort(nodes.begin(),nodes.end(),predica);
     vector<Host*>::iterator i;
     for(i=nodes.begin();(n>0) && (i!=nodes.end());n--,i++){
        res.push_back((*i)->getHostname());
    //    cout << (*i)->getHostname() << " " << (*i)->getCpu() << endl;
        delete *i;
     }
     for(;i!=nodes.end();++i)
        delete *i;

     ldap_unbind(ldap_server);
     globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);

} /* get_ldap_attribute */
}
```

SmallBlue example

This is a sample game playing program.

SmallBlue.C (standalone version)

```
#include <string>
#include <iostream>
#include<fstream>

using namespace std;

#include"GAME.C"

int Simulate(GAME newgame, int col) {
   int l=0,s;
   int start=newgame.Value(col,WHITE);
   newgame.Play(col,WHITE);
// newgame.Inverse();
   l=0;
   for(int k=1;k!=XSIZE+1;k++) {
      s=newgame.Value(k,BLACK);
      if (s>l)
         l=s;
   };
   start-=l;
   return start;
}

main() {
   GAME Current(XSIZE,YSIZE);
   int s,l,k,toplay;
   char c[2];
   while (true) {
      Current.Display();
      do {
         cout << "?";
         cin >> c;
         c[1]='\0';
         l=atoi(c);
      } while ((l<1) || (l>XSIZE) || !Current.CanPlay(l) );
      Current.Play(l,BLACK);
      if (Current.HasWon(l,BLACK)) {
         Current.Display();
         exit(1);
      };
      l=-100000;
      for(k=1;k!=XSIZE+1;k++) {
         if (Current.CanPlay(k)) {
```

```
            s=Simulate(Current,k);
            if (s>l) {
               l=s;
               toplay=k;
            };
         };
      };
      if (l==-100000) {
         cout << "NULL" << endl;
         exit(1);
      };
      Current.Play(toplay,WHITE);
      if (Current.HasWon(toplay,WHITE)) {
         Current.Display();
         exit(1);
      };
   };
};
```

SmallBlueMaster.C

```
#define _GNU_SOURCE // mandatory to use get_current_dir_name
#include <unistd.h>
#include "globus_common.h"
#include "globus_gass_server_ez.h"
#include "itso_gram_jobs_callback.h"
#include "itso_globus_ftp_client.h"
#include <vector>
#include <iostream>
#include "broker.h"
#include "itso_gass_server.h"
#include "GAME.C"

using namespace itso_gass_server;
using namespace std;

main() {
   //Start a GASS server locally that will listen on 10000 port
   //all the results of the evualtion.  We will stop it at the end
   //It cannot be defined as a standalone class because the static callback
   //does not take any argument.  So it is impossible afterwards in the
   //callback to refer to the object.
   StartGASSServer(10001);

   // ITSO_GRAM_CLIENT does not start the module
   // lets do it
   if (globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE) != GLOBUS_SUCCESS)
   {
      cerr << " Cannot start GRAM module";
```

```
      exit(2);
   };

   // the game
   GAME Current(XSIZE,YSIZE);
   // used to temporary store columns positions, evaluation results
   int s,l,k,toplay;
   // used to store human inputs
   char c[2];

   // The node vector should be initialized with the value of the nodes
   // what is missing here is the globus calls to the globus MDS server
   // to get these values.  So for the exercise, you can use grid-info-search
   // to find 8 hosts on which you can submit your queries.
   vector<string> node;

   // ask the broker to find 8 nodes
   itso_broker::GetLinuxNodes(node,8);

   // variable used in all for loops
   int i;

   // Here we want test the existence of the file as there is
   // no such checking in the ITSO_GLOBUS_FTP_CLIENT class
   FILE*    fd = fopen("SmallBlueSlave","r");
        if(fd == NULL)
   {
              printf("Error opening local smallblueslave");
              exit(2);
   }
   else {
      //that fine, lets go for FTP
      // we can close fd descriptor because a new one
      // will be opened for each ITSO_GLOBUS_FTP_CLIENT object
      fclose(fd);
      // the ITSO_CB callback object is used to determine
      // when the transfer has been completed
      vector<ITSO_GLOBUS_FTP_CLIENT*> transfer;
      //never forget to activate the Globus module you want to use
      globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);

      // 8 transfer, let create 8 locks
      string dst;
      for(i=0;i!=8;i++) {
         cout << node[i] << endl;
         dst="gsiftp://"+node[i]+"/~/SmallBlueSlave";
         transfer.push_back(new
ITSO_GLOBUS_FTP_CLIENT("SmallBlueSlave",const_cast<char*>(dst.c_str())));
      };
```

```
      // Let s begin the transfer in parallel (in asynchronous mode)
      for(i=0;i!=8;i++)
         transfer[i]->StartTransfer();
      // Let wait for the end of all of them
      for(i=0;i!=8;i++)
         transfer[i]->Wait();
      globus_module_deactivate(GLOBUS_FTP_CLIENT_MODULE);
   };

   // get the hostname using the globus shell function
   // instead of POSIX system calls.
   char hostname[MAXHOSTNAMELEN];
   globus_libc_gethostname(hostname, MAXHOSTNAMELEN);

   // used to store the RSL commands.
   string rsl_req;

   //Start the GRAM callback server
   ITSO_GRAM_JOBS_CALLBACK callback_server;
   //create all the jobs objects that will be used to submit the
   //requests to the nodes.  We use a vector to store them.
   vector<ITSO_GRAM_JOB*> job;
   for(i=0;i!=8;i++) {
      job.push_back(new ITSO_GRAM_JOB(&callback_server));
   };

   // By using gridftp SmallBlueSlave is copied onto the rmeote hosts
   // as a plain file.  needs to chmod +x to make it executable
   // otherwise the job submission will fail
   cout << "chmod +x on the remote hosts to make SmallBlueSlave executable" <<
endl;
   for(i=0;i!=8;i++) {
      rsl_req = "&(executable=/bin/chmod) (count=1) (arguments= \"+x\"
SmallBlueSlave)";
      if ( job[i]->Submit(node[i],rsl_req))
         exit(1);
   }
   for(i=0;i!=8;i++)
      job[i]->Wait();


   while (true) {
      Current.Display();
      do {
         cout << "?";
         cin >> c;
         c[1]='\0';
         l=atoi(c);
      } while ((l<1) || (l>XSIZE) || !Current.CanPlay(l) );
```

```
Current.Play(l,BLACK);
if (Current.HasWon(l,BLACK)) {
   Current.Display();
   break;
};
// Serialize to disk the Current variable
// so that it could be used by the GRAM
// subsystem and transferred on the remote execution
// nodes
Current.ToDisk("GAME");

Current.Display();
cout << endl;

// remove eval file for each new jobs submission
// otherwise results will be appended to the same files
unlink("eval");

for(i=0;i!=8;i++) {
   cout << "submission on " << node[i] << endl;;
   char tmpc[2];
   sprintf(tmpc,"%d",i);
   // build the RSL commands
   rsl_req = "&(executable=SmallBlueSlave) (arguments=";
   rsl_req+= tmpc[0];
   rsl_req+= ") (stdout=https://";
   rsl_req +=hostname;
   rsl_req +=":10001";
   rsl_req +=get_current_dir_name();
   rsl_req +="/eval) (stdin=https://";
   rsl_req +=hostname;
   rsl_req +=":10001";
   rsl_req +=get_current_dir_name();
   rsl_req +="/GAME) (count=1)";
   // submit it to the GRAM
   if (Current.CanPlay(i))
      if (job[i]->Submit(node[i],rsl_req))
         exit(1);
};
// And Wait
for(i=0;i!=8;i++)
   if (Current.CanPlay(i))
      job[i]->Wait();

// worse case :-)
l=-100000;

//Here we are reading the eval files.  All the jobs
//has been completed so we should have all the results
```

```
      //in the eval file
      ifstream results("eval");
      while (!results.eof()) {
         results >> k >> s;
         // get the best one
         if (s>l) {
            l=s;      // store its value
            toplay=k; //remember the column to play
         };
      };
      results.close();

      // nothing in the file, that means we cannot play
      // so it is NULL
      if (l==-100000) {
         cout << "NULL" << endl;
         break;
      };

      // AI plays here and checks if it won
      Current.Play(toplay,WHITE);
      if (Current.HasWon(toplay,WHITE)) {
         Current.Display();
         break;
      };
   };

   //finished stop everything
   StopGASSServer();
}
```

SmallBlueSlave.C

```
#include <iostream>

using namespace std;
#include "GAME.C"

main(int arc, char** argv) {
   GAME Current(XSIZE,YSIZE);
   Current.FromDisk();
   //which column should we simulate ?
   int col=atoi(argv[1]);
   int start=Current.Value(col,WHITE);
   Current.Play(col,WHITE);

   int l=0,s,k;
   for(k=1;k!=XSIZE+1;k++) {
      s=Current.Value(k,BLACK);
```

```
      if (s>l)
         l=s;
   };
   start-=l;

   // send back the information to the server
   cout << col << " " << start << endl;
};
```

GAME Class

```
#include <string>
#include <iostream>
#include<fstream>
#define HOWMANY 4
#define WHITE 1
#define BLACK HOWMANY+1
#define XSIZE 8
#define YSIZE 10

using namespace std;


class GAME {
       char* data;
       int xsize;
       int ysize;
       int get(int& i,int& j) {
              return data[i-1+(j-1)*xsize];
       };
       int get1(int& i) { return data[i-1];};
       void set(int& i,int& j,int who) { data[i-1+(j-1)*xsize]=(char)who; };
       int available(int col) {
          if (get1(col)!=0)
             return 0;
          int j;
          for(j=ysize;get(col,j)!=0;--j) {};
          return j;
       };
       bool test(int i,int j,int& col, int& line,int& player) {
          int k,x,y;
          for(int decalage=0;decalage!=HOWMANY;decalage++) {
             int r=0;
             for(k=HOWMANY-1;k!=-1;k--) {
                x=i*(k-decalage)+col;
                y=line+j*(k-decalage);
                if ((x<1) || (x>xsize) || (y>ysize) || (y<1) ) {
                   break;
                };
```

```
            if (player==get(x,y))
               r+=1;
         };
         if (r==HOWMANY)
            return true;
      };
      return false;
   };
   int calculate(int i,int j,int& col, int& line) {
      int res=0,k=0,x,y;
      for(int decalage=0;decalage!=HOWMANY;decalage++) {
         int r=0;
         for(k=HOWMANY-1;k!=-1;k--) {
            x=i*(k-decalage)+col;
            y=line+j*(k-decalage);
            if ((x<1) || (x>xsize) || (y>ysize) || (y<1) ) {
               r=0;
               break;
            };
            r+=get(x,y);
         };
         // Special
         if (r==HOWMANY) {
            r+=2000;
         }
         else if (r==HOWMANY*(BLACK)) {
            r+=1000; }
         res+=r;
      };
      return res;
   };
public:
   GAME(GAME& newgame) {
      xsize=newgame.getxsize();
      ysize=newgame.getysize();
      data=(char*)calloc(xsize*ysize,1);
      memcpy(data,newgame.getData(),xsize*ysize);
   }
   GAME(int x,int y)
   : xsize(x),ysize(y)
   {
      data=(char*)calloc(x*y,1);
   };
   ~GAME() {
           free(data);
   };
   char* getData() { return data;};
   int getxsize() { return xsize;};
   int getysize() { return ysize;};
```

```
void ToDisk(string filename) {
   ofstream out(filename.c_str());
   int j,i;
   for(j=1;j<=ysize;++j)
     for(i=1;i<=xsize;++i)
      out << get(i,j) << " ";
   out.close();

};
void FromDisk() {
   int j,i,valeur;
   for(j=1;j<=ysize;++j)
     for(i=1;i<=xsize;++i) {
      cin >> valeur;
           set(i,j,valeur);
     };
};
void Inverse() {
   int i,j;
   for(j=1;j<=ysize;++j)
     for(i=1;i<=xsize;++i)
      if (get(i,j)==WHITE)
         set(i,j,BLACK);
      else if (get(i,j)==BLACK)
         set(i,j,WHITE);
};
void Display() {
   int j,i;
   cout << endl << endl;
   cout << "----------";
   cout << endl;
   for(j=1;j<=ysize;++j) {
     cout << "|" ;
     for(i=1;i<=xsize;++i)
      if (get(i,j)==WHITE)
         cout << 'O';
      else if (get(i,j)==BLACK)
         cout << 'I';
      else
         cout << ' ';
      cout << "|" ;
      cout << endl;
   };
   cout << "----------" << endl;
   cout << " 12345678";
};
bool Play(int col,int who) {
   int j;
   if (get1(col)!=0)
```

```
            {
               cout << "NO !!!!!!!!!!!!";
               cout << col; exit(1);
            };
            for(j=ysize;get(col,j)!=0;--j) {};
            set(col,j,who);
         };
         GAME& operator=(GAME& G2) {
            memcpy(data,G2.getData(),xsize*ysize);
         };
         bool CanPlay(int& col) {
            if (get1(col)!=0)
               return false;
            return true;
         };
         bool HasWon(int col, int player) {
            int line;
            bool res=false;
            line=available(col);
            line++;
            res|=test(-1,0,col,line,player);
            res|=test(-1,-1,col,line,player);
            res|=test(0,-1,col,line,player);
            res|=test(1,-1,col,line,player);
            return res;
         };
         int Value(int col, int player) {
            int line;

            if ((line=available(col))==0)
               return 0;
            set(col,line,player);

            int res=0;
            res+=calculate(-1,0,col,line);
            res+=calculate(-1,-1,col,line);
            res+=calculate(0,-1,col,line);
            res+=calculate(1,-1,col,line);
            set(col,line,0);
            return res;
         };
};
```

Makefile

```
#globus-makefile-header --flavor=gcc32 globus_io globus_gss_assist
globus_ftp_client globus_ftp_control globus_gram_job globus_common
globus_gram_client globus_gass_server_ez > globus_header
```

```
include globus_header

all:  SmallBlueSlave SmallBlueMaster SmallBlue

%.o: %.C
   g++ -c $(GLOBUS_CPPFLAGS) $< -o $@

SmallBlue:SmallBlue.o GAME.o
   g++ -o $@ -g $^

SmallBlueSlave:SmallBlueSlave.o GAME.o
   g++ -o $@ -g $^

SmallBlueMaster: GAME.C SmallBlueMaster.C itso_gram_jobs_callback.C itso_cb.C
itso_globus_ftp_client.C itso_gass_server.C broker.C
   g++ -g -o $@ $(GLOBUS_CPPFLAGS) $(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS)
-lldap_gcc32pthr
```

HelloWorld example

Below is a sample HelloWorld program.

HelloFrontEnd.C

```
#define _GNU_SOURCE // mandatory to use get_current_dir_name
#include <unistd.h>
#include “globus_common.h”
#include “itso_gram_job.h”
#include “itso_gass_copy.h”
#include <iostream>
#include <fstream>
#include “itso_gass_server.C”
#include <cc++/socket.h>
#include <cstdlib>

#ifdef  CCXX_NAMESPACES
using namespace std;
using namespace ost;
#endif

// static variables used by the FrontEndServer threads.
// They are initialized at startup and readonly.
// Note that for a real production case, ticket will probably
// variable.  A locking mechanism with mutex needs to be
// used.
static char hostname[MAXHOSTNAMELEN];
static long ticket;
```

```
//*************************************************************
//  We use here the GNU Common C++ classes to implement the
//  basic server that will listen onb port 4096 to manages client
//  request and redirects them on the application server
//  The Server class is derived from TCPSocket class
//  Each client Session object (Thread) is from class TCPSession
//  The member run() is executed for each acceped connection.
// and http://www.gnu.org/software/commonc++/docs/refman/html/classes.html
//  for details
//*************************************************************
//*************************************************************
class GridApp : public TCPSocket
{
protected:
        bool onAccept(const InetHostAddress &ia, tpport_t port);

public:
        GridApp(InetAddress &ia);
};

GridApp::GridApp(InetAddress &ia) : TCPSocket(ia, 4096) {};

bool GridApp::onAccept(const InetHostAddress &ia, tpport_t port)
{
        cout << “accepting from: “ << ia.getHostname() << “:” << port << endl;;
        return true;
}

class GridAppSession : public TCPSession
{
private:
        void run(void);
        void final(void);

public:
        GridAppSession(TCPSocket &server);
};

GridAppSession::GridAppSession(TCPSocket &server) :
TCPSession(server)
{
        cout << “creating session client object” << endl;
};

void GridAppSession::run(void)
{
   string node;
```

```
   node = "t0";
   // node = getNode(MDS, SLA, Workload, ...) in a real production
   // case.  For simplicity here, we use a fixed address.

        InetAddress addr = getPeer();
        *tcp() << "welcome to " << addr.getHostname() << endl;
        *tcp() << "ticket: " << ticket << endl;
        *tcp() << "hostname: " << node << endl;
   endSocket();
}

void GridAppSession::final(void)
{
        delete this;
}

//*************************************************************
main() {
   // We start here the GASS server that will be used:
   // - to transfer the Compile script to the application nodes
   //  to perform the library compilation
   // - to transfer the HelloServer used on the remote hosts to
   // manage clients requests
   // The GASS server arbitraly listens on port 20000
   StartGASSServer(20000);
   // get the hostname using the globus function
   globus_libc_gethostname(hostname, MAXHOSTNAMELEN);

   // ITSO_GRAM_CLIENT does not start the module
   // lets do it
   if (globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE) != GLOBUS_SUCCESS)
   {
      cerr << " Cannot start GRAM module";
      exit(2);
   };

   string node;

   // we use a fixed address for this simple example but
   // nodes=getNodes(SLA, MDS, Workload, ...) in a more complex example
   // we should get here the list of nodes where the application is
   // supposed to run.
   node = "t0";

   // variable used in all for loops
   int i;

   // Here we want test the existence of the file as there is
   // no such checking in the ITSO_GLOBUS_FTP_CLIENT class
```

```
FILE*    fd = fopen("commoncpp2-1.0.8.tar.gz","r");
     if(fd == NULL)
{
           printf("Error opening commoncpp2-1.0.8.tar.gz file");
           exit(2);
}
else
    {
   //that fine, lets go for FTP
   // we can close fd descriptor because a new one
   // will be opened for each ITSO_GLOBUS_FTP_CLIENT object
   fclose(fd);
   //never forget to activate the Globus module you want to use
   globus_module_activate(GLOBUS_GASS_COPY_MODULE);
   //************************************************************
   // In this section we perform the transfer of the dynamic library
   // to the "application server".
   // We use the ITSO_GLOBUS_FTP_CLIENT class to do the task
   // Note that we use the local directory to perform the task.
   // In a real case example, a storage server would be used
   // instead of a local directory =>
   // globus-url-copy gsiftp://storage_server/commoncpp2-1.0.8.tar.gz
   //       gsiftp://application_server/commoncpp2-1.0.8.tar.gz
   //************************************************************
   GLOBUS_URL source,destination;
            source.setURL("gsiftp://m0/tmp/commoncpp2-1.0.8.tar.gz");

            string dst;
            dst="gsiftp://"+node+"/~/commoncpp2-1.0.8.tar.gz";
            destination.setURL(dst);

            globus_module_activate(GLOBUS_GASS_COPY_MODULE);
            ITSO_GASS_TRANSFER transfer;
            transfer.Transfer(source,destination);
            transfer.Wait();
            globus_module_deactivate(GLOBUS_GASS_COPY_MODULE);


   //************************************************************
   // In this section we submit the compilation of the
   // dynamic libray.  The script Compile that must be in the
   // current directory is transferred to the remote host
   // and executed.  The result is the installation
   // of the CommonC++ toolkit in the tmp directory of the
   // user under which the script was executed.
   // (globus in the lab environement of the redbook)
   //************************************************************
   // used to store the RSL commands.
   string rsl_req;
```

```
      ITSO_GRAM_JOB job;

      cout << "library compilation on " << node << endl;;
      rsl_req = "&(executable=https://";
      rsl_req +=hostname;
      rsl_req += ":20000";
      rsl_req +=get_current_dir_name();
           rsl_req += "/Compile) (count=1)";
      // submit it to the GRAM
      cout << rsl_req << endl;
      //job.Submit(node,rsl_req);
      //job.Wait();
      //*************************************************************

      // ticket=getTicket(time, IP address, SLA, ...) in a
      // real production case
      ticket=random();

      // node=getNodes(SLA, MDS, Workload, Application Type, ...)
      // in a real production case
      node="t0";

      ITSO_GRAM_JOB job2;

      cout << "start app server on " << node << endl;;
      rsl_req = "&(executable=https://";
      rsl_req +=hostname;
      rsl_req += ":20000";
      rsl_req +=get_current_dir_name();
      rsl_req += "/HelloServer) (environment=(LD_LIBRARY_PATH $(HOME)/tmp/lib)
) (count=1) (arguments=";
      char tmpstr[20];
      sprintf(tmpstr,"%ld ",ticket);
      rsl_req += tmpstr;
      rsl_req += node;
      rsl_req += ")";
      // submit it to the GRAM
      cout << rsl_req << endl;
      job2.Submit(node,rsl_req);
      job2.Wait();

      // Front End server startup
      // We use the GNU CommonC++ classes to implement a
      // very basic server.  See below for explanation
      // and
http://www.gnu.org/software/commonc++/docs/refman/html/classes.html
      // for details
      GridAppSession *tcp;
```

```
            BroadcastAddress addr;
         addr = "localhost";
            cout << "binding for: " << addr.getHostname() << ":" << 4096 <<
endl;

         GridApp server(addr);

            while(server.isPendingConnection(300000))  // the server runs for
                        // a limited period of time
            {
                   tcp = new GridAppSession(server);
                   tcp->detach();  // the new thread is daemonize to manage
                   // the client connection
       };

   };

   // Stop the GASS server when exiting the program
   StopGASSServer();
}
```

HelloServer.C

```
#include <cc++/socket.h>
#include <cstdlib>

#ifdef  CCXX_NAMESPACES
using namespace std;
using namespace ost;
#endif

class GridApp : public TCPSocket
{
public:
         GridApp(InetAddress &ia);
   void end();
};

GridApp::GridApp(InetAddress &ia) : TCPSocket(ia, 4097) {};

void GridApp::end() { endSocket(); };

int main(int argc,char** argv )
{
tcpstream tcp;
    long ticket;
    ticket=atol(argv[1]);

    InetAddress addr;
```

```
    addr = argv[2];
         cout << "addr: " << addr << ":" << 4097 << endl;
         GridApp server(addr);
    long i;
    // daemonize
    long l =fork();
    if (l>0)
       exit(0); // parent exits here
    while(server.isPendingConnection(300000))
         {
                    tcp.open(server);
                    if(tcp.isPending(Socket::pendingInput, 2000))
                    {
                         tcp >> i;
                         cout << "user entered " << i << ticket << endl;
          if (i!=ticket)
             tcp << "Bad ticket" << endl;
          else
             tcp << "Hello World !";
                    }
                    cout << "exiting now" << endl;
                    tcp.close();
    };
};
```

HelloClient.C

```
#include <cc++/socket.h>
#include <string>
using namespace std;
using namespace ost;
// g++ -g -I/usr/include/cc++2 -L/usr/lib -o S Ser2.C -lccgnu2  -lpthread -ldl

class GridApp : public TCPStream {
       char line[200];
   public:
       GridApp(InetHostAddress &ia) : TCPStream(ia, 4097) {};
       char* readline() {
          tcp()->getline(line,200);
          return line;
       }
};

int main(int argc,char** argv) {
   InetHostAddress FrontEndServerAddress,AppServerAddress;
   FrontEndServerAddress=argv[1];

   TCPStream str(FrontEndServerAddress,4096);
```

```
   string hostname,ticket;
   str >> ticket;
   str >> ticket;
   str >> ticket;
   str >> ticket;
   str >> ticket;
   cout << "Ticket:" << ticket << endl;
   str >> hostname;
   str >> hostname;
   str >> hostname;
   cout << "Hostname:" << hostname << endl;

   AppServerAddress=hostname.c_str();
   GridApp App(AppServerAddress);

   App << ticket << endl;

   cout << App.readline();
};
```

Makefile

```
globus-makefile-header --flavor=gcc32 globus_io globus_gass_copy
globus_gss_assist globus_ftp_client globus_ftp_control globus_gram_job
globus_common globus_gram_client globus_gass_server_ez > globus_header

include globus_header

all:  HelloServer HelloClient HelloFrontEnd

%.o: %.C
   g++ -c $(GLOBUS_CPPFLAGS) $< -o $@

HelloServer: HelloServer.C
   g++ -g -I/usr/local/include/cc++2 -L/usr/local/lib -o $@ $^ -lccgnu2
-lpthread -ldl

HelloClient: HelloClient.C
   g++ -g -I/usr/local/include/cc++2 -L/usr/local/lib -o $@ $^ -lccgnu2
-lpthread -ldl

HelloFrontEnd: HelloFrontEnd.C  itso_gram_job.C itso_cb.C itso_gass_copy.C
itso_gass_server.C
   g++ -o $@ -I/usr/local/include/cc++2 -L/usr/local/lib $(GLOBUS_CPPFLAGS)
$(GLOBUS_LDFLAGS) $^ $(GLOBUS_PKG_LIBS) -lccgnu2  -lpthread -ldl
```

Lottery example

Below is a sample for emulating a lottery. To compile GenerateDraws.C issue:

```
g++ -o GenerateDraws -O 3 GenerateDraws.C
```

GenerateDraws.C

```
#include <cstdlib>
#include <unistd.h>
#include <list>
#include <algorithm>
#include <fstream>

int GetARandomNumberBetween1And60() {
   returnrandom()/(RAND_MAX/60)+1;
};

void initrandom(unsigned seed){
   #define RANDSIZE 256
   static char randstate[RANDSIZE];
   initstate(seed, randstate, RANDSIZE);
}

main(int argc,char ** argv) {
   initrandom(getpid());  // seeding for a new sequence of pseudo-random
integers
   // let generate 8 number between 1 and 60
   list<long int> Series;
   long int n;
   long long DrawNumber,InitialDrawNumber;
   ofstream OutputFileMonitor;
   OutputFileMonitor.open(“Monitor”);

   DrawNumber=atoll(argv[1]);
   InitialDrawNumber=DrawNumber;

   do {
      int i;
      Series.clear();
      for(i=8;i;i--) {
         //find 8 different number between 1 and 60
         do
            n=GetARandomNumberBetween1And60();
         while (find(Series.begin(),Series.end(),n)!=Series.end());
         Series.push_front(n);
      };

      //let display the result
```

```
      Series.sort();
      list<long int>::iterator j;
      for(j=Series.begin();j!=Series.end();++j)
         cout << *j << " ";
      cout << endl;

      OutputFileMonitor.seekp(1,ios::beg);
      OutputFileMonitor <<
100*(InitialDrawNumber-DrawNumber)/InitialDrawNumber;
   } while (--DrawNumber);

   OutputFileMonitor.seekp(1,ios::beg);
   OutputFileMonitor << "100";
   OutputFileMonitor.close();
}
```

Submit script

```
#the script takes the tested draw as parameter
#example: ./Submit 3 4 5 32 34 43
n=1000000
NodesNumber=12

#temporary working directory on the execution nodes
TMP=.$HOSTNAME

i=0
#the loop variable is used is all the "for" loops
#the format is 1 2 3 4 .... n
loop=""
# use here the broker developped for the redbook
# see chapter 8 (mds executable)
for node in $(mds $NodesNumber | xargs)
do
   Nodes[$i]=$node
   loop=${loop}" "${i}
   i=$(( $i + 1 ))
done

echo The number of draws tested is $n
a=$*
#sort the numbers in the specified draw
# 2 45 23 12 32 43 becomes 2 12 23 32 43 45 so that we could use
# grep to test this draw and the ouput of the draw programs.
param=$(echo $a | tr " " "\n" | sort -n | xargs )

# parrarell transfer of the draw executable
# we submit jobs in the background, get their process id
# and uses the wait command to wait for their completion
```

```
# this method is also used for the jobs submission
echo Transferring executable files
for i in $loop
do
   gsissh -p 24 ${Nodes[$i]} “[ -d $TMP ] || mkdir $TMP” &
   ProcessID[$i]=$!
done
for i in $loop
do
   wait ${ProcessID[$i]}
   gsiscp -P 24 GenerateDraws ${Nodes[$i]}:$TMP  &
   ProcessID[$i]=$!
done
for i in $loop
do
   wait ${ProcessID[$i]}
   gsissh -p 24 ${Nodes[$i]} “chmod +x ./$TMP/GenerateDraws” &
   ProcessID[$i]=$!
done
#file should be made executable
#on all the execution nodes
echo Jobs submission to the grid
for i in $loop
do
   wait ${ProcessID[$i]}
   echo ${Nodes[$i]}
   EXE=”cd $TMP;./GenerateDraws $n | grep “‘””$param’” && echo GOT IT on
$HOSTNAME’
   gsissh -p 24 ${Nodes[$i]} “$EXE” &
   ProcessID[$i]=$!
done

#for monitoring, we copy locally the Monitor files
# created on each compute nodes.  This file content the
# percentage of tested draws.  Each files is suffixes by
# the nodes number.  $statusnum is actually the sum of all
# the percentage (Monitor files) devided by 100.  When it
# equals the number of nodes, that means that we have finished
echo Monitoring
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
   sleep 5  #we poll every 5 seconds
   statussum=0
   for i in $loop
   do
      gsiscp -q -P 24 ${Nodes[$i]}:$TMP/Monitor Monitor.$i
      status=$(cat Monitor.$i)
```

```
      statussum=$((  $status + $statussum   ))
      echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
   done
   statussum=$((  $statussum  / 100 ))
done
#cleanup the tmp directory
for i in $loop
do
   wait ${ProcessID[$i]}
   gsissh -p 24 ${Nodes[$i]} “rm -fr ./.$TMP” &
   ProcessID[$i]=$!
done
```

GenerateDrawsGlobus.C

To compile this program, just issue:

```
g++ -o GenerateDrawsGlobus GenerateDrawsGlobus.C

#include <cstdlib>
#include <unistd.h>
#include <list>
#include <algorithm>
#include <string>
#include <fstream>

int GetARandomNumberBetween1And60() {
   returnrandom()/(RAND_MAX/60)+1;
};

void initrandom(unsigned seed){
   #define RANDSIZE 256
   static char randstate[RANDSIZE];
   initstate(seed, randstate, RANDSIZE);
}

// the first argument is the hostname provided by SubmitGlobus script
// the second argument is the number of draws that must be generated
main(int argc,char ** argv) {
   initrandom(getpid());  // seeding for a new sequence of pseudo-random
integers
   // let generate 8 number between 1 and 60
   list<long int> Series;
   long int n;
   long long DrawNumber,InitialDrawNumber;
   ofstream OutputFileMonitor;
   string filename;
   filename=”Monitor.”;
   filename.append(argv[1]);
   OutputFileMonitor.open(filename.c_str());
```

```
   DrawNumber=atoll(argv[2]);
   InitialDrawNumber=DrawNumber;

   do {
      int i;
      Series.clear();
      for(i=8;i;i--) {
         //find 8 different number between 1 and 60
         do
            n=GetARandomNumberBetween1And60();
         while (find(Series.begin(),Series.end(),n)!=Series.end());
         Series.push_front(n);
      };
      //let display the result
      Series.sort();
      list<long int>::iterator j;
      for(j=Series.begin();j!=Series.end();++j)
         cout << *j << " ";
      cout << endl;

      OutputFileMonitor.seekp(1,ios::beg);
      OutputFileMonitor <<
100*(InitialDrawNumber-DrawNumber)/InitialDrawNumber;
   } while (--DrawNumber);

   OutputFileMonitor.seekp(1,ios::beg);
   OutputFileMonitor << "100";
   OutputFileMonitor.close();
}
```

SubmitGlobus script

```
#the script takes the tested draw as parameter
#example: ./Submit 3 4 5 32 34 43
n=10000000
NodesNumber=8

#temporary filename used by by GenerateDrawsGlobus
#to monitor the job
#we can also use the process id to increase the granularity
TMP=$HOSTNAME

i=0
#the loop variable is used is all the "for" loops
#the format is 1 2 3 4 .... n
loop=""
# use here the broker developped for the redbook
# see chapter 8 (mds executable)
```

```
for node in $(mds $NodesNumber | xargs)
do
   Nodes[$i]=$node
   loop=${loop}" "${i}
   i=$(( $i + 1 ))
done

echo The number of draws tested is $n
a=$*
#sort the numbers in the specified draw
# 2 45 23 12 32 43 becomes 2 12 23 32 43 45 so that we could use
# grep to test this draw and the ouput of the draw programs.
param=$(echo $a | tr " " "\n" | sort -n | xargs )

#Start the gass server on each nodes
# clean up the Monitoring file when leaving
for i in $loop
do
  rsl='&(executable=$(GLOBUS_LOCATION)/bin/globus-gass-server)(arguments=-c -t
-r)(environment=(LD_LIBRARY_PATH
$(GLOBUS_LOCATION)/lib))(file_clean_up=Monitor.'"$TMP)"
   globusrun -o -r ${Nodes[$i]} "$rsl" > gass-server.$i &
done
#file should be made executable
#on all the execution nodes
echo Jobs submission to the grid
rsl="+"
for i in $loop
do
   echo ${Nodes[$i]}
   rsl=${rsl}"(&(resourceManagerContact=\"${Nodes[$i]}\")"

rsl=${rsl}"(executable=\$(GLOBUSRUN_GASS_URL)$PWD/GenerateDrawsGlobus.sh)(argum
ents=$TMP $n
\"$param\")(subjobStartType=loose-barrier)(file_stage_in=(\$(GLOBUSRUN_GASS_URL
)$PWD/GenerateDrawsGlobus
GenerateDrawsGlobus.$TMP))(file_clean_up=GenerateDrawsGlobus.$TMP)(environment=
(LD_LIBRARY_PATH \$(GLOBUS_LOCATION)/lib)) )"
done
echo $rsl
globusrun -s -o "$rsl" &
#for monitoring, we copy locally the Monitor files
# created on each compute nodes.  This file content the
# percentage of tested draws.  Each files is suffixes by
# the nodes number.  $statusnum is actually the sum of all
# the percentage (Monitor files) devided by 100.  When it
# equals the number of nodes, that means that we have finished

echo Monitoring
```

```
rm -f Monitor.*
statussum=0
while (( $statussum != $NodesNumber ))
do
   echo
   sleep 5  #we poll every 5 seconds
   statussum=0
   for i in $loop
   do
      if [ -s gass-server.$i ]
      then
      contact=$(cat gass-server.$i)
           globus-url-copy $contact/~/Monitor.$TMP file://$PWD/Monitor.$i
           status=$(cat Monitor.$i)
           statussum=$((  $status + $statussum   ))
           echo ${Nodes[$i]}:Monitor $(cat Monitor.$i) %
      fi
   done
   statussum=$((  $statussum  / 100 ))
done

#Stop the gassserver
for i in $loop
do
   contact=$(cat gass-server.$i)
   globus-gass-server-shutdown $contact
done
```

C/C++ simple examples

Below are more simple examples.

gassserver.C

```
#include “globus_common.h”
#include “globus_gass_server_ez.h”
#include <iostream>
#include “itso_cb.h”

ITSO_CB callback; //invoked when client wants to shutdown the server

void callback_c_function() {
   callback.setDone();
}

main() {
```

```
// Never forget to activate GLOBUS module
globus_module_activate(GLOBUS_GASS_SERVER_EZ_MODULE);

// let s define options for our GASS server
unsigned long server_ez_opts=0UL;

//Files openfor writing will be written a line at a time
//so multiple writers can access them safely.
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_LINE_BUFFER;

//URLs that have ~ character, will be expanded to the home
//directory of the user who is running the server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_EXPAND;

//URLs that have ~user character, will be expanded to the home
//directory of the user on the server machine
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_TILDE_USER_EXPAND;

//"get" requests will be fullfilled
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_READ_ENABLE;

//"put" requests will be fullfilled
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_WRITE_ENABLE;

// for put requets on /dev/stdout will be redirected to the standard
// output stream of the gass server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDOUT_ENABLE;

// for put requets on /dev/stderr will be redirected to the standard
// output stream of the gass server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_STDERR_ENABLE;

// "put requests" to the URL https://host/dev/globus_gass_client_shutdown
// will cause the callback function to be called.  this allows
// the GASS client to communicate shutdown requests to the server
server_ez_opts |= GLOBUS_GASS_SERVER_EZ_CLIENT_SHUTDOWN_ENABLE;

// Secure
char* scheme="https";
//unsecure
//char* scheme="http";
     globus_gass_transfer_listenerattr_t  attr;
     globus_gass_transfer_listenerattr_init(&attr, scheme);

     //we want to listen on post 10000
globus_gass_transfer_listenerattr_set_port(&attr, 10000);

//Now, we can start this gass server !
 globus_gass_transfer_listener_t    listener;
```

```
        globus_gass_transfer_requestattr_t  * reqattr      = GLOBUS_NULL;
//purpose unknown

   int err = globus_gass_server_ez_init(&listener,
                                        &attr,
                                        scheme,
                                        GLOBUS_NULL, //purpose unknown
                                        server_ez_opts,
                callback_c_function); //or GLOBUS_NULL otherwise
                //GLOBUS_NULL); //or GLOBUS_NULL otherwise

   if((err != GLOBUS_SUCCESS)) {
           cerr << "Error: initializing GASS (" << err << ")" << endl;
          exit(1);
   }

        char *
gass_server_url=globus_gass_transfer_listener_get_base_url(listener);
   cout << "we are listening on " << gass_server_url << endl;

        //wait until it is finished !
   //that means that the "put requests" to the URL
https://host/dev/globus_gass_client_shutdown
   //ITSO_CB implements the symchronization mechanism by using a mutex
   //and a condition variable
        callback.Wait();  // shutdown callback

   //stop everything
   globus_gass_server_ez_shutdown(listener);
   globus_module_deactivate(GLOBUS_GASS_SERVER_EZ_MODULE);
}
```

Checking credentials

Here is a quick example of how to check if your credentials are valid in a C or C++ program. These credentials are generated via the **globus-proxy-init** shell command.

```
#include "globus_gss_assist.h"
#include <iostream>
main() {
   gss_cred_id_t credential_handle = GSS_C_NO_CREDENTIAL;

   OM_uint32 major_status;
   OM_uint32 minor_status;

   major_status = globus_gss_assist_acquire_cred(&minor_status,
                               GSS_C_INITIATE, /* or GSS_C_ACCEPT */
                               &credential_handle);
```

```
        if (major_status != GSS_S_COMPLETE)
       cout << “unable to authenticate !” << endl;
   else
       cout << “that s fine” << endl;
}
```

Submitting a job

Here is a small C example that provides a skeleton for submitting a job in a C program. The examples in this publication use the ITSO_GRAM_JOB C++ class, which is basically a C++ wrapper to this C skeleton.

```
#include <stdio.h>
#include “globus_gram_client.h”
#include <globus_gram_protocol_constants.h>

/* It is the function used when the remote
 * Job Manager needs to contact your local program to inform it
 * about the status of the remote  program. It is passed along
 * with the the job_request to the remote computer
 */

static void callback_func(void * user_callback_arg,
                          char * job_contact,
                          int state,
                          int errorcode);

/* Setting up the GRAM monitor. The monitor will stall
 * this program until the remote program is terminated,
 * either through failure or naturally. Without the monitor,
 * this program would submit the job, and end before the job
 * completed. The monitor works with a lock. Only one function
 * may access the Done flag at a time, so in order to access it,
 * the gram must set a lock on the monitor variable, so that
 * nothing else may access it, then change it, and finally
 * unlock it. This is seen later in the code.
 */

/* This whole structure is the monitor */

typedef struct
{
    globus_mutex_t mutex;
    globus_cond_t cond;
    globus_bool_t done;
} my_monitor_t;

/*************************************************
```

```
                    Main Code
************************************************/

int main(int argc, char ** argv)
{
    int callback_fd;
    int job_state_mask;
    int rc; /* The return value of the request function.
             * If successful, it should be 0 */

    char * callback_contact; /* This is the identifier for
                              * the callback, returned by
                              * globus_gram_job_request
                              */

   char * job_contact; /* This is the identifier for the job,
                        * returned by globus_gram_job_request
                        */

   char * rm_contact;
   char * specification;
   globus_bool_t done;
   my_monitor_t Monitor;

    /* Retrieve relevant parameters from the command line */

    if (argc!= 3 && argc != 4 && argc != 5)
    {
        /* invalid parameters passed */
        printf("Usage: %s <rm_contact> <specification> "
                        "<job_state_mask> <-debug>\n",
                argv[0]);
        exit(1);
    }

   if ((rc = globus_module_activate(GLOBUS_GRAM_CLIENT_MODULE))
   != GLOBUS_SUCCESS)
   {
   printf("\tERROR: gram module activation failed\n");
   exit(1);
   }

   rm_contact = (char *)globus_malloc(strlen(argv[1])+1);
   strcpy(rm_contact, argv[1]);
   specification = (char *)globus_malloc(strlen(argv[2])+1);
   strcpy(specification, argv[2]);
   if (argc > 3)
      job_state_mask = atoi(argv[3]);
   else
```

```
   job_state_mask = GLOBUS_GRAM_PROTOCOL_JOB_STATE_ALL;

/* Initialize the monitor function to look for callbacks.  It
 * initializes the locking mechanism, and then the condition
 * variable
 */
globus_mutex_init(&Monitor.mutex, (globus_mutexattr_t *) NULL);
globus_cond_init(&Monitor.cond, (globus_condattr_t *) NULL);

/* entering the monitor and clearing the flag. Locking the
 * Monitor to prevent anything else from changing the value of
 * Monitor.done
 */
globus_mutex_lock(&Monitor.mutex);

/* Change the value of Monitor.done to false, initializing it */
Monitor.done = GLOBUS_FALSE;

/* Releasing the lock on the monitor, letting anything else access it */
globus_mutex_unlock(&Monitor.mutex);

/* Setting up the communications port for returning the callback.
 * You pass it the callback function.  The callback_contact is the
 * callback identifier returned by the function
 */

 globus_gram_client_callback_allow(callback_func,
                   (void *) &Monitor,
                   &callback_contact);

 printf("\n\tTEST: submitting to resource manager...\n");

 /* Send the GRAM request.  The rm_contact, specification, and
  * job_state_mask were retrieved earlier from the command line
  * The callback_contact was just returned by
  * globus_gram_client_callback_allow.  The job_request is returned by
  * this function
  */

 rc = globus_gram_client_job_request(rm_contact,
                                     specification,
                                     job_state_mask,
                                     callback_contact,
                                     &job_contact);

 if (rc != 0) /* if there is an error */
 {
     printf("TEST: gram error: %d - %s\n",
            rc,
```

```
                /* translate the error into english */
                globus_gram_client_error_string(rc));
         exit(1);
    }

#ifdef CANCEL
    sleep(3);
    printf("\tTEST: sending cancel to job manager...\n");

    if ((rc = globus_gram_client_job_cancel(job_contact)) != 0)
    {
       printf("\tTEST: Failed to cancel job.\n");
       printf("\tTEST: gram error: %d - %s\n",
               rc,
               globus_gram_client_error_string(rc));
       return(1);
    }
    else
    {
       printf("\tTEST: job cancel was successful.\n");
    }
#endif

/* Wait until there is a callback saying there was a termination, either
 * successful or failed.  We lock the Monitor again so as to ensure that
 * no one else tampers with it. Then we wait until the condition is
 * signaled by the callback_function. When it is signaled, and
 * Monitor.done is set to GRAM_TRUE - (these two things always happen
 * in conjunction in our callback_func) Then we unlock the monitor and
 * continue the program.
 */

    globus_mutex_lock(&Monitor.mutex);
    while (!Monitor.done)
    {
       /* Within the cond_wait function, it unlocks the monitor,
        * allowing the callback_func to take the lock. When it gets a
        * cond_signal, it re-locks the monitor, and returns to this
        * program.  But DO NOT unlock the monitor yourself- use the
        * globus_gram_cond_wait function, as it insures safe
        * unlocking.
        */
        globus_cond_wait(&Monitor.cond, &Monitor.mutex);
    } /* endwhile */

    globus_mutex_unlock(&Monitor.mutex);

    /* Remove Monitor.  Given that we are done with our monitor, (it has
     * already held the program until the job completed) we can now dispose
```

```
     * of it. We destroy both the mutex and the condition.  This frees up any
     * space it may have occupied.
     */

    globus_mutex_destroy(&Monitor.mutex);
    globus_cond_destroy(&Monitor.cond);

    /* Free up the resources of the job_contact, as the job is over, and
     * the contact is now useless.
     */
    globus_gram_client_job_contact_free(job_contact);

    /* Deactivate GRAM */
    globus_module_deactivate(GLOBUS_GRAM_CLIENT_MODULE);
}
/******************************************************************
 * This is the callback function, as per the definition. We can  write
 * whatever we want into the function, but remember that the
 * cond_signal must be triggered and Monitor.done must also be set  to
 * true to exit the waiting loop in the main code. The function is called
 * from the job manager, which provides values for state and errorcode
 */

void callback_func(void * user_callback_arg,
                   char * job_contact,
                   int state,
                   int errorcode)
{
    my_monitor_t * Monitor = (my_monitor_t *) user_callback_arg;

    switch(state)
    {
    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING:
          printf("\tTEST: Got GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING"
                   " from job manager\n");
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE:
           printf("\tTEST: Got GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE"
                   " from job manager\n");
          break; /* Reports state change to the user */

    case GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED:
          printf("\tTEST: Got GLOBUS_GRAM_PROTOCOL_JOB_STATE_FAILED"
                   " from job manager\n");
          globus_mutex_lock(&Monitor->mutex);
          Monitor->done = GLOBUS_TRUE;
          globus_cond_signal(&Monitor->cond);
          globus_mutex_unlock(&Monitor->mutex);
```

```
            break; /* Reports state change to the user */
        case GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE:
            printf("\tTEST: Got GLOBUS_GRAM_PROTOCOL_JOB_STATE_DONE"
                   " from job manager\n");
            globus_mutex_lock(&Monitor->mutex);
            Monitor->done = GLOBUS_TRUE;
            globus_cond_signal(&Monitor->cond);
            globus_mutex_unlock(&Monitor->mutex);
            break; /* Reports state change to the user */
        }
    }
```

C

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246936

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG243936.

© Copyright IBM Corp. 2003. All rights reserved.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
README.6936	A sort desrciption of the following files
3936Samp.zip	A zip file including various source files from examples in this publication
3936Samp.tar	A tar file including the same contents as the above zip file

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip or untar the contents of the Web material zip/tar file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 372. Note that some of the documents referenced here may be available in softcopy only.

- *Introducing IBM Tivoli License Manager*, SG24-6888
- *Introduction to Grid Computing with Globus,* SG24-6895
- *Fundamentals of Grid Computing,* REDP3613

Other publications

These publications are also relevant as further information sources:

- *The Java CoG Kit User Manaul*, Gregor von Laszewski, Beulah Alunkal, Kaitzar Amin, Jarek Gawor, Mihael Hategan, Sandeep Nijsure. Available at:

  http://www.globus.org/cog/manual-user.pdf
- *Network Security Essentials: Application and Standards*. Stallings, W. (2000). Prentice-Hall Inc.
- *A standard for architecture description*, by R. Youngs, D. Redmond-Pyle, P. Spaas and E. Kahan. IBM Systems Journal, Volume 38, Number 1, 1999 Enterprise Solutions Structure available at:

  http://www.research.ibm.com/journal/sj/381/youngs.html
- *On Death, Taxes, and the Convergence of Peer-toPeer and Grid Computing,* by Ian Foster, Adriana Iamnitchi:

  http://people.cs.uchicago.edu/~anda/papers/foster_grid_vs_p2p.pdf
- Foster, et al, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, ISBN 1558604758
- *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, found at:

  http://www.globus.org/research/papers/anatomy.pdf

© Copyright IBM Corp. 2003. All rights reserved.

- *A Brief Introduction to Grid Technology*, found at:

  http://www.bo.infn.it/alice/introgrd/introgrd/
- *Computational Grids*, found at:

  http://www.globus.org/research/papers/chapter2.pdf
- *The Globus Project: A Status Report*, found at:

  ftp://ftp.globus.org/pub/globus/papers/globus-hcw98.pdf
- *GridFTP Update January 2002*, found at:

  http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf
- *Grid Service Specification*, found at:

  http://www.gridforum.org/ogsi-wg/drafts/GS_Spec_draft03_2002-07-17.pdf
- *Internet Draft GridFTP: Protocol Extensions to FTP for the Grid*, found at:

  http://www-fp.mcs.anl.gov/dsl/GridFTP-Protocol-RFC-Draft.pdf
- *Internet Draft Internet X.509 Public Key Infrastructure Proxy Certificate Profile*, found at:

  http://www.ietf.org/internet-drafts/draft-ietf-pkix-proxy-03.txt
- *MDS 2.2 User's Guide*, found at:

  http://www.globus.org/mds/mdsusersguide.pdf
- *The Open Grid Services Architecture and Data Grids*, found at:

  http://aspen.ucs.indiana.edu/CCPEwebresource/c600gridkunszt/c600FINAL
- *GridServices_DataGridv3.pdf The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, found at:

  http://www.globus.org/research/papers/ogsa.pdf
- *A Resource Management Architecture for Metacomputing Systems*, found at:

  ftp://ftp.globus.org/pub/globus/papers/gram97.pdf
- *RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, found at:

  http://www.ietf.org/rfc/rfc3280.txt
- *Web Services Conceptual Architecture (WSCA 1.0)*, found at:

  http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf
- *Web Service Description Language (WSDL) 1.1*, found at:

  http://www.w3.org/TR/wsdl

- *A Security Architecture for Computational Grids*. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.

  ftp://ftp.globus.org/pub/globus/papers/security.pdf

- *Managing Security in High-Performance Distributed Computing*. I. Foster, N. T. Karonis, C. Kesselman, S. Tuecke. Cluster Computing, 1(1):95-107, 1998.

  ftp://ftp.globus.org/pub/globus/papers/cc-security.pdf

Online resources

These Web sites and URLs are also relevant as further information sources:

- Globus

  http://www.globus.org

- GSI-Enabled OpenSSH, visit

  http://grid.ncsa.uiuc.edu/ssh/

- Globus GASS

  http://www-fp.globus.org/gass/

- BSD Licensing

  http://www.opensource.org/licenses/bsd-license.php

- MIT Licensing

  http://opensource.org/licenses/gpl-license.php

- Apache Software license

  http://www.opensource.org/licenses/apachepl.php

- Open source Initiative license information

  http://opensource.org/docs/certification_mark.php

- Lesser General Public License

  http://www.opensource.org/licenses/lgpl-license.php

- GNU Public License

  http://www.gnu.org/copyleft/gpl.html

- IBM Public License

  http://www.opensource.org/licenses/ibmpl.php.

- FLEXlm supported license models

  http://www.globetrotter.com/flexlm/lmmodels.sthm

- FLEXlm general information

  http://www.globetrotter.com/flexlm/flexlm.shtm
- Tivoli License Manager

  http://www.ibm.com/software/tivoli/products/license-mgr/
- IBM License Use Management

  http://www.ibm.com/software/is/lum/
- Platofrm Global License Broker

  http://www.platform.com/products/wm/glb/index.asp
- Globus Commodity Grid toolkts (CoGs)

  http://www.globus.org/cog/
- Grid Computing Environment (GCE)

  http://www.globus.org/research/development-environments.html
- Grid Enabled MPI

  http://www.niu.edu/mpi/
- Grid Application Development software (GrADS)

  http://nhse2.cs.rise.edu/grads/
- IBM Grid Toolbox

  http://www.alphaworks.ibm.com/tech/gridtoolbox
- Grid Application Framework for Java

  http://www.alphaworks.ibm.com/tech/GAF4J
- IBM Data Management products

  http://www.ibm.com/software/data/
- Database Access and Integration Services Working group

  http://www.gridforum.org/6_DATA/dais.htm
- MDS information provider example

  http://www-unix.mcs.anl.gov/~slang/mds_iprovider_example/
- OpenSSH

  http://www.openssh.org
- NFS Version 4 Open Source Reference Implementation

  http://www.citi.umich.edu/projects/nfsv4
- Avaki

  http://www.avaki.com

- Global File System

  http://www.sistina.com/products_gfs.htm
- Replica Location Service

  http://www.globus.org/rls
- Replica Location Service

  http://www.globus.org/rls
- GDMP

  http://project-gdmp.web.cern.ch/project-gdmp
- OGSA-DAI

  http://umbriel.dcs.gla.ac.uk/NeSC/general/projects/OGSA_DAI/
- gSOAP

  http://www.cs.fsu.edu/~engelen/soap.html
- Spitfire project

  http://spitfire.web.cern.ch/Spitfire/
- Storage Resource Broker

  http://www.npaci.edu/DICE/SRB/
- European Union data grid

  http://www.eu-datagrid.org
- Griphyn Project

  http://www.griphyn.org/
- Particle Physics Data Grid

  http://www.ppdg.net/
- Grid Portal Development Kit

  http://doesciencegrid.org/projects/GPDK/
- GSI-OpenSSH

  http://www.nsf-middleware.org/NMIR2/
- Grid Portal Development Kit

  http://doesciencegrid.org/projects/GPDK/
- CommonC++

  http://www.gnu.org/software/commonc++/
- gSOAP Plugin

  http://gsoap2.sourceforge.net

- Platform Computing

  http://www.platform.com

- DataSynapse

  http://www.datasynapse.com

- United Devices

  http://www.ud.com

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Index

A
Access Control System 88
accounting 5, 11
all permissive license 63
Apache 63
application architecture considerations 43
application development 66
Application Development Environments 66
application flow 45, 52
Application Programming Interfaces 5
application server 55
application status 234
asynchronous calls 117
authentication 14–15
authorization 14–15
Avaki 92

B
bandwidth 77
blocking calls 117
broker 8, 23, 33, 123–124
  considerations 33
broker example 127, 327
browing MDS with a GUI 127
BSD 63

C
C bindings 5
C considerations 53
C++ 105
C++ examples 305
C/C++ 66
caching 87
callback 109, 250, 315
cancel job 235
CAS 88
CCA 67
Certificate Authority 84
CertUtils 136
chargeback 5, 11
checkpoint-restart 38, 56
Chimera 103
CICS 58
CoG 5, 54, 66
cog.properties file 135
CoGs 134
command line interface 57
Common Component Architecture 67
CommonC++ library 282
Community Authorization Service 88
comodity grid kit (CoG) 66
compiler settings 55
computational grids 3
condition variables 106
Condor-G 29–30
consumers 46
CORBA 5, 66
credentials 7, 137

D
data caches 73
data encryption 17, 84
data grid 4, 39–40
data input/output 57
data management 5, 9, 14, 25, 75, 163
  considerations 28
data producer and consumer 46
data store 57
data topology 40
data topology graph 78
data types 79
Database Access and Integration (DAI) 99
Database Access and Integration Services 86
DFS 91
digital certificates 16
distinguished name 15
distributed by design 67
distributed by nature 67
DLLs 55
dumprsl 21, 113
DUROC 18, 21
dynamic libraries depedencies 281
Dynamically-Updated Request Online Coallocator
  see DUROC

© Copyright IBM Corp. 2003. All rights reserved.

E

encrypted sockets 171
European DataGrid Project 102
extended block mode 202

F

factories 292
Federated Databases 99
federated databases 74
file staging 167
flavors 107
FLEXlm 64

G

GAF4J 67
GASS 26–27, 134, 163, 178, 257
  API 155, 178
  example - Java 155
  server 113
gatekeeper 18, 21
GCE 66
General Public License 64
GFS 95
GIIS 23, 25, 125
GIS 14, 23
Globus cache management 192
Globus common 106
Globus module 109
Globus Project 4
Globus Toolkit
  contents 5
  overview 4
  Version 2.2 4
  Version 3 4–5, 289
globus_gass_server_ez API 188
globus_gass_transfer API 187
globus_gram_client API 118
globus_io 169
globus_module_activate 109
globus_module_deactivate 109
globus-job-run 21, 111
globus-job-submit 112
globus-makefile-header 108
globusrun 18–19, 113
globus-url-copy 26
GPFS 91
GPL 64
GrADS 67
GRAM 18, 27, 105, 134, 138, 290
GRAM example - Java 139
Graphical Tools 127
grid
  components 6
  computational 3
  data 4
  infrastructure considerations 13
  overview 3
  scavenging 4
  types 3
Grid Access to Secondary Storage
  see GASS
Grid Application Development Software 67
Grid Application Examples 246
Grid Application Framework for Java 67
grid applications 45
Grid File Transfer Protocol
  see GridFTP
Grid Index Information Service
  see GIIS
Grid Information Service
  see GIS
Grid Resource Allocation Manager
  see GRAM
Grid Resource Information Service
  see GRIS
Grid Security Infrastructure
  see GSI
grid types
  data grid 40
GridFTP 9, 26, 134, 195, 290
  client-server 151
  example - Java 148
grid-info-search 125
grid-proxy-init 15, 134
GriPhyn Project 102
GRIS 24–25
GSI 7, 14–15, 26, 170, 290
GSI socket 173
GSI-Enabled OpenSSH 15–16, 53, 246, 369
gSOAP 291

H

hardware devices 55
header files 107
heartbeat monitoring 38
Hello World example 278, 341

HTML 58
HTTP 58

I
IBM Grid Toolbox 67
ID administration 16
ID mapping 17
index service 293
information providers 23, 25
information services 5, 14, 22, 290
  considerations 25
infrastructure components 14
infrastructure considerations 13
instances 292
interactive jobs 53
interconnect 5
inter-process communication 22, 33, 56
intra-grids 3
introduction 1
ITSO broker example 327
ITSO_CB 268
ITSO_CB (callback) example 315
ITSO_GASS Transfer example 306
ITSO_GLOBUS_FTP_CLIENT example 311
ITSO_GRAM_JOB 118
ITSO_GRAM_JOB example 316
ITSO_GRAM_JOB object 119

J
Java 5, 53, 66, 133, 232
JavaCoG 133
JNDI 141
job 45
job criteria 51
job dependencies 54
job flow 11, 45
job management 10, 22
job manager 18, 21
job scheduler 9
job submission 117–118, 123, 251
job topology 56
jobs 45
JVM 40

K
Kerberos 26
knock-out criteria 68

L
LD_LIBRARY_PATH 260, 282
LDAP 8, 23, 67, 128, 143
LDAP query 125
ldapsearch 25, 125–126
ldd command 108
LDIF 290
Lesser General Public License 64
LGPL 64
libc 106
libraries 108
license agreement 62
License Broker 66
license management 64
License Manager 65
license models 62
License Use Management 65
life-cycle management 5–6
load balancing 31
  considerations 32
locking 75
locking management 106
loose coupling 49
Lottery example 349

M
Makefile 106
Makefile example 108
MCAT 101
MDS 8, 14, 22, 25, 33, 105, 134
MDS example - Java 141
memory size 55
Message Passing Interface 66
message queues 57
meta scheduler 9
metadata 94, 101
mirroring 87
MIT 63
mixed platform environments 40
modules 107
Monitoring and Discovery Service
  see MDS
MPI 66
MPICH-G2 34, 66
mutex 109, 201
mutex life-cycle 106

N

naming 292
Network File System
  see NFS
network file systems 26
network topology 39
networked flow 46, 49
NFS 91
non-blocking calls 117
non-functional requirements 35

O

OGSA 3, 5, 11, 33, 54
OGSI 6, 11, 58, 290
Open Grid Service Architecture 292
Open Grid Services Architecturer
  see OGSA
open source licensing 63
open source software 63
OpenSSH 15
OpenSSL 7, 16, 170
operating systems 54
OSI 64
overview of grid 3

P

pache 63
parallel application flow 46–47
parallel applications 52
parallelism 82, 202
parallelization 48
partial file transfer 26, 201
Particle Physics Data Grid 103
peer to peer 88
peer-to-peer computing 4
performance 36
Perl 53, 66
persistent license 64
persistent storage 38
Petascale Virtual Data Grid 102
ping 122
Platform Global License Broker 66
Platform Globsal License Broker 66
portal 6, 34, 215
  considerations 35
  integration with application 232
portal example 216
portal login flow 219
portal security 233
portType 295
POSIX 169
power grid 3
producers 46
programming environment 106
programming language considerations 53
programming model 5
proxy 15, 134
proxy certificate 7
proxyType variable 136
Public Key Infrastructure 84
Puissance 4 262
Python 5, 66

Q

qualification scheme 69, 298

R

Redbooks Web site 372
  Contact us xvi
reliability 37
reliable data transfer 26
Reliable File Transfer Service 296
Replica Catalog 96
Replica Location Service 97, 296
replica management 28, 208
replica management installation 211
replication 86
resource management 5, 10, 14, 17
  considerations 22
resources
  grid-wide 24
  local 24
RSL 18, 113, 120, 134, 139, 145, 165
  example - Java 145
  for data management 165
runtime considerations 40
runtime environment 55

S

SAL 63
sandbox 89, 250
scavenging grids 4
scheduler 9, 21, 29
  considerations 30
secure communication 16

Secure Socket Layer 16
security 5, 7, 14
  considerations 17
serial flow 46–47
service data browser 293
service provider license agreement 63
service providers 63
servlet 218
shared data access 74
signal management 106
single sign-on 14, 17
Small Blue example 262, 331
SOAP 6, 58
SOAP message security 290
software license 62
Spitfire 100
SRB 101
SSL/TSL 16
StartGASSServer example 324
StopGASSServer example 324
Storage Area Networks 26
Storage Resource Broker 100
storage servers 26
Storage Tank 94
stream mode 202
striped data transfer 26
subjobs 22, 50
submitting a Job 110
Subscriber Access Licenses 63
system management 38
system return value 57

T

temporary data spaces 76
third-party file transfer 26, 149
thread life cycle management 106
thread-safety 109
time sensitive data 77
Tivoli License Manager 65
topology considerations 38
transaction 58
transfer agent 88
types of grids 3

U

URLCopy 154
usability requirements 59

V

viral license 64
virtual computer 3
Virtual Data Toolkit 103

W

web services 5, 66
WebSphere 7, 33–34
wrapper script 111
WSDL 292

X

XML 6, 58

Enabling Applications for Grid Computing with Globus

Enabling Applications for Grid Computing with Globus

Enable your applications for grid computing

Utilize the Globus Toolkit

Programming hints, tips, and examples

This IBM Redbook, a follow-on to Introduction to *Grid Computing with Globus*, SG24-6895, discusses the issues and considerations for enabling an application to run in a grid environment. Programming examples are provided based on the Globus Toolkit V2.2.

The first part of this publication addresses various considerations related to grid-enabling an application, from the perspective of the infrastructure, the application, and the data requirements.

The second part of this publication provides many programming examples in C/C++ and Java to help solidify the concepts of grid computing and the types of programming tasks that must be handled when developing an application intended to run in a grid environment

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6936-00 ISBN 0738453331